European Commission

JRC TECHNICAL REPORT

Web Application Programming Interfaces (APIs): general-purpose standards, terms and European Commission initiatives

APIs4DGov study — digital government APIs: the road to value-added open API-driven services

Santoro, M., Vaccari, L., Mavridis, D., Smith, R. S., Posada, M., Gattwinkel, D.

2019

Joint Research Centre

EUR 29984 EN

Contact information
Name: European Commission, Joint Research Centre (JRC), Digital Economy Unit (JRC.B6)
Address: Via Enrico Fermi, 2749 — 21027 Ispra (VA), Italy
Email: lorenzino.vaccari@ec.europa.eu
Tel.: +39 0332 78 57 58

EU Science Hub
https://ec.europa.eu/jrc

JRC118082

EUR 29984 EN

PDF ISBN 978-92-76-13183-0 ISSN 1831-9424 doi:10.2760/675

Luxembourg: Publications Office of the European Union, 2019

How to cite this report: Santoro, M., Vaccari, L., Mavridis, D., Smith, R., Posada M. and Gattwinkel D., *Web APIs: general-purpose standards, terms and European Commission initiatives*, EUR 29984 EN, Publications Office of the European Union, Luxembourg, 2019, ISBN 978-92-76-13183-0, doi:10.2760/675, JRC118082.

How to cite the list of standards on which this report is based on:
Vaccari, L. and Santoro, M., *API standards and technical specifications - APIs4DGov*, European Commission, Joint Research Centre (JRC), 2019 [Dataset] PID: http://data.europa.eu/89h/5a431f38-1e2c-449a-898e-34f2a3234c3b.

Contents

Preface

This technical report has been prepared as part of the study 'APIs4DGov —digital government APIs: the road to value-added open API-driven services' ([1]) and will contribute to the study's final outputs in early 2020. Performed by the Joint Research Centre (JRC) of the European Commission, in collaboration with the Directorate-General (DG) for Communications Networks, Content and Technology, the study will last for 2 years and is being conducted in the context of the European Commission's Digital Single Market strategy. The work aims to improve the understanding of the current use of APIs in digital government and their added value, as well as to assess the feasibility of establishing a European API framework for digital government.

The study addresses many different aspects of this topic, with the focus of this report on web API standards and being directed mainly at professional practitioners, providers, consumers and technical users working within the API digital universe. For this audience, the list of web API standards and technical specifications on which we have based our analysis must be considered as an integral part of this report. Because we want the list to be a live document and a single source of truth, we have published a dynamic table to the JRC data catalogue rather than a fixed table in the appendix. The introductory sections, the glossary, and the list of European initiatives will be of interest to a broader audience; particularly those working on API adoption in organisations who need to ensure interoperability issues are addressed.

As an interim output of the study, this report should be useful for supporting the technical implementation of APIs. Based on workshops, interviews and meetings (as well as online fora) held as part of the study, involving both private and public sector actors, data on specific subjects such as API-related concepts, terminology, standards and European Commission related initiatives have been gathered and reviewed. It has been challenging to develop such an understanding of these topics and answering questions such as 'What is an API?', 'What is REST?' 'What is a web service and how is it different from an API?' and 'How can APIs be secured?', as well as 'What are the main ways to document and design an API?'.

This technical report, therefore, also aims to clarify and disseminate our understanding of crucial API subjects, namely API-related concepts, terms, standards and technical specifications. It is important to note that the documents examined do not include domain-specific standards or technical specifications. In addition, we aimed to be as neutral as possible (i.e. by not expressing preferences for a certain technical specification or standard) in the selection of and definitions used in relation to the topics presented. Equally, we do not give specific recommendations or details of trends in relation to how to adopt APIs in organisations, an area we aim to address in another part of the study. To allow all readers to find out more about their topics of interest, we have paid particular attention to providing references and links for each of the concepts, terms and standards presented.

Finally, our intention is that readers will contribute to further work in this area. While explaining each subject, we tried to be as precise and rigorous as possible, but we know that some issues remain unclear. For example, several definitions exist for the same subject and some terms are associated with different definitions. Both of these issues should be resolved by further work. For this reason, we encourage readers to provide us, through the contact email address provided, with their insights and feedback. This will both improve our knowledge and contribute to the final report of the APIs4DGov study.

([1]) Application Programming Interface: The calls, subroutines, or software interrupts that comprise a documented interface so that an application program can use the services and functions of another application, operating system, network operating system, driver, or other lower-level software program (Shnier, 1996).

Acknowledgements

Several people contributed to this report. We would like to thank current and former colleagues of the JRC.B6 Digital Economy Unit, including Anders Friis-Christensen (prior project leader of the study), Stefano Nativi for his insightful comments during the preparation of this work and colleagues from JRC.B6, namely Alex Kotsev, Alessandro Dalla Benetta, Michele Rovera and Andrea Perego, for their comments on the content of the document. We also would like to thank all our colleagues at the European Commission who participated in the fruitful cross-DG workshops, discussions and meetings, in particular Pedro Obando and Serge Novaretti (DG Communications Networks, Content and Technology) and Tanya Chetcuti (DG Informatics).

We would also like to thank the following project advisory board members for their advice on this document: Patrick Amarelis (Enterprise Architect at Direction interministérielle du numérique et du système d'information et de communication de l'État), David Berlind (editor in chief of ProgrammableWeb.com), Marco Combetto (innovation manager at Trentino Digitale S.p.A.), Georges Lobo (programme manager at DG Informatics) and Roberto Polli (full stack developer of the Italian government's Digital Transformation Team).

We would also like to thank ProgrammableWeb, a world-leading source of news and information about internet-based APIs, in particular David Berlind, its editor in chief, for providing API directory data for the study. Some of the current trends discussed in this report are based on these data.

All errors remain the sole responsibility of the authors.

Authors

Mattia Santoro (PhD) is a researcher at the Earth and Space Science Informatics Laboratory of the National Research Council in Italy. His main research interests are semantic discovery and the interoperability of environmental models. He is responsible for the GEO Discovery and Access Broker operational environment.

Lorenzino Vaccari (PhD) is a computer scientist and scientific officer at the Digital Economy Unit of the JRC in Ispra, Italy. Lorenzino leads the APIs4DGov study and his specialties include APIs, blockchain, open data and digital technologies supporting the evolution of digital governments.

Dimitrios Mavridis (PhD) is a research economist at the Digital Economy Unit at the JRC in Seville, Spain. He is part of the project team of the APIs4DGov study that investigates the added value of open government data through APIs, among other issues. His current research interests include the economic analysis of interoperability, platform economics and the effects of information systems on governance.

Robin Smith (PhD) is a scientific officer at the Digital Economy Unit of the JRC in Ispra, Italy. He is part of the APIs4DGov team, the infrastructure for spatial information in Europe (Inspire) coordination team and the team responsible for the ELISE (enabling digital government through geospatial and location intelligence) action of the interoperability solutions for public administrations, businesses and citizens (ISA^2) programme. Robin has a background in ecology and town and regional planning, and his research over the last 20 years has focused on the role of information and communications technology (ICT) in public participation, collaborative e-science and the development of European spatial data infrastructures from both social and technical perspectives. He is currently exploring the role of geospatial intelligence in the digital transformation of government.

Monica Posada (MSc) is a scientific officer at the Digital Economy Unit of the JRC in Ispra, Italy. Monica is part of the APIs4DGov team and her work focuses on the analysis of the techno-economic impacts of the pervasive uptake of information technologies in the digital era. Her professional interests span various aspects of data science, specifically the application of advanced data analysis and visualisation techniques for the design of decision support systems.

Dietmar Gattwinkel (MSc) is a seconded national expert with the eGovernment and Trust Unit (Unit H.4) of the DG Communications Networks, Content and Technology, European Commission, Luxembourg. He is the business manager of the APIs4DGov project and was also the project leader of an open data initiative for the state enterprise Saxon Computer Science Services in Saxony, Germany.

Abstract

From their inception, digital technologies have had a huge impact on our society. In both the private and the public sectors, they have contributed to, or at times driven, change in organisational structures, ways of working, and how products and services are shaped and shared. Governments and public administration units, driven by the digital evolution of information and communications technology (ICT), are evolving from traditional workflow-based public service provisions to digital equivalents, with more innovative forms of government and administration looking for the engagement of citizens and the private sector to co-create final services through user-centric approaches. Application Programming Interfaces (APIs) have contributed to this notable shift in the adoption of technology, especially when used over the web. They have affected the global economy of the private sector and are contributing to the digital transformation of governments.

To explore this in more detail, the European Commission recently started the APIs4DGov study. One of the outputs of the study is an analysis of the API technological landscape, including its related standards and technical specifications for general purpose use. The goal of the analysis presented in this brief report is to support the definition of stable APIs for digital government services adopted by governments or single public administration units. Such adoption would avoid the need to develop ad hoc solutions that could have limited scalability or potential for reuse. Instead, the work suggests that we should consider a number of existing standards provided by standardisation bodies or, at least, technical specifications written by well-recognised consortia, vendors or users.

The aim of this report is also to support API stakeholders in the identification and selection of such solutions. To do this, it first gives a series of definitions to help the reader understand some basic concepts, as well as related standards and technical specifications. Then, it presents the description and classification (by resource representation, security, usability, test, performance and licence) of the standards and technical specifications collected. A shortlist of these documents (based on their utilisation, maintenance and stability) is also proposed, together with a brief description of each of them. Also, a set of API related European Commission initiatives is presented. Finally, the report provides a useful glossary with definitions of the relevant terms collected within the APIs4DGov study.

1 Introduction

Digital technologies and related solutions are changing every dimension of our life, including our interaction with government authorities and public services. This digital transformation has an important impact on citizens, the private sector and public administration units acting in different policy areas. Within the public sector, digital transformation designates the process of the conversion or substitution of analogue public administration operations to their digital counterparts. Information and Communications Technology (ICT) provides tools for the faster and more efficient processing of data within public administration units; more efficient public services can result in significant cost savings or the development of new kinds of services for the same cost.

The public sector in the European Union has advanced a lot in terms of e-government and digital transformation initiatives during the last 20 years, both in developing examples on an individual level and in their integration across borders and sectors (European Commission, 2019a), especially through the related interchange of data between administrations (IDA) (European Commission, 2000), interoperable delivery of European eGovernment services to public administrations, business and citizens (IDABC) (European Commission, 2004), and interoperability solutions for public administrations, businesses and citizens (ISA and ISA^2) programmes (European Commission, 2017a). The European Commission has encouraged in many ways and for a considerable period the development of e-government and digital government across the EU (OECD, 2019). It provides guidance and suggests voluntary measures to foster the development of digital government in addition to having adopted legislation in specific domains, such as on eInvoicing (European Union, 2014a) and the Open Data and Public Sector Information (PSI) Directive (European Union, 2019), as well as technical infrastructure components such as the Regulation on electronic identification and trust services-eIDAS (European Union, 2014b), helping to further the use of electronic identification (eID) across Europe.

As ICT allows the faster and more efficient processing of data, increased efficiency may lead to cost savings or the development of new kinds of services. It can be seen that application programming interfaces (APIs) have become a foundational technological component of modern digital architectures, affecting every sector of the global economy. Although not novel as a technological solution, their capability to enable machine-to-machine communication make them instrumental for a functional digital fabric for both the private and the public sectors. Moreover, they can be considered key enablers of the accelerated evolution of governments and their agencies, from analogue (manual, paper) operations and workflows to operations and workflows being implemented by digital artefacts, making them faster, more flexible and interoperable.

For the digital sector, especially in relation to the web, the existence of proper standards and specifications guarantees interoperability among countless digital assets as defined by a number of standardisation bodies and communities, including the Internet Engineering Task Force (IETF), the World Wide Web Consortium (W3C), the International Organization for Standardization (ISO), the European Committee for Standardization (CEN), the European Telecommunications Standards Institute (ETSI) and the Institute of Electrical and Electronics Engineers (IEEE). To support the definition of stable APIs for digital government services, there is a need to avoid the adoption of ad hoc solutions and, instead, to rely on a number of existing standards provided by standardisation bodies. Where a suitable standard is not available, technical specifications written by well-recognised consortia, vendors or users may be important for developments in relation to interoperability objectives.

This document analyses the current situation in terms of API standards and technical specifications, offering practitioners and developers a quick reference document. Moreover, it focuses on cross-domain subjects, i.e. it does not include domain-specific standards (health, geospatial, internet of things — IoT, etc). The outputs presented in this document will contribute to the technological landscape description objective of the study. Launched in 2018, the 2-year study is being undertaken by the Joint Research Centre (JRC) of the European Commission and the Directorate-General (DG) for Communications Networks, Content and Technology. It aims to improve the understanding of the current use of APIs, especially web APIs, in digital government, to examine their added value in such contexts and to assess the feasibility of establishing a European API framework for digital government (European Commission, 2018a).

The document is divided into three parts. The rest of this section presents the definitions of an API and the main related concepts. It also presents two main methods for the evaluation of the maturity of API adoption by an organisation. The second part describes the documents collected so far, the criteria adopted for their selection, the rationale for their classification and some simple statistics about them. The entire list of documents is published in the JRC Data Catalogue (Vaccari and Santoro, 2019). Notice that the list does not

include documents which are considered of (i) general purpose for the Web and (ii) consolidated background knowledge of the reader (e.g. HTTP, JSON, XML, URI, SOA, ROA, RDF, etc.). The second part also provides a shortlist of technical specifications and standards selected as most relevant for the reader and for the study. It also describes a series of European Commission initiatives that can be related to standards and technical specifications. In the third part of the document, we provide a glossary of terms relevant to the study. This list complements the main definitions given below, and references are provided to guide the reader to relevant sources of information.

1.1 Scope of the report

APIs are a flexible and lightweight approach that can be used by an organisation to provide software programming functionalities to internal and third-party applications. Because of the extreme variability and rapid evolution of ICT and web technologies, particularly in the API landscape, any list of existing APIs in relevant sectors or in relation to themes relevant for digital government would most likely rapidly become obsolete. In addition, many repositories, such as ProgrammableWeb.com (ProgrammableWeb.com, 2019) and RapidAPI (RapidAPI, 2019), already provide and update such lists.

Thus, in this document, rather than focusing on current technological trends and domain-specific standards, such as those described by the Open Geospatial Consortium (OGC, 2019a) and (HL7.org, 2019), the aim is to propose some relevant general purpose standards and technical specifications for web APIs, i.e. the kind of APIs that operate over the web.

This document does not provide any recommendations as far as the use of any particular technical specification or standard. Moreover, the aim of this report was not to collect every available standard but to compile an evolving list and provide information on updates to web API standards. If the reader is searching for this list for the web, he or she can check the excellent work described in (Wilde, 2018) and maintained in (Wilde, 2019).

Instead, we present the description and classification (by resource representation, security, usability, test, performance and licence) of the standards and technical specifications collected. The goal of the shortlist presented in the document is to give the reader basic information about a selected number of technical specifications and standards that have been found to support the study and/or to be of particular (real or potential) importance for governments.

1.2 Definitions

This section illustrates the main concepts that have to be taken into account when dealing with technical design and evaluation of the maturity level of API implementation. As shown in the next sections, in the design phase, it useful to compare two types of web interface, namely web APIs and Web Services. Moreover, depending on the type of client-server coupling, the software designer can choose from two main approaches: Remote Procedure Call (RPC) or REpresentational State Transfer (REST). Next in the section, we present how API implementation can be evaluated by using two kinds of model: the first assesses the level of access to the shared resources and a second evaluates the degree of compliance with the REST architectural style. Finally, we briefly define what microservices are, as they are an important and emergent architectural solution that could be adopted behind the implementation of APIs.

1.2.1 Application Programming Interfaces

The API concept is not new. It probably first appeared in 1968 defined as 'a collection of code routines that provide external users with data and data functionality' (Cotton and Greatorex, 1968). Because APIs are general technological solutions, they can be used for many purposes. Thus, we can adopt a more recent and extended APIs explanation that defines them as 'the calls, subroutines, or software interrupts that comprise a documented interface so that an application program can use the services and functions of another application, operating system, network operating system, driver, or other lower-level software program' (Shnier, 1996).

From a software engineering point of view, APIs constitute the interfaces of the various building blocks that a developer can assemble to create an application. An application developer utilises APIs to build an application by combining various available software libraries to achieve a specific goal. While the notion of programmatic interfaces as a collection of methods exported by a certain code library is not new, with the advent of the web, and in particular Web 2.0, the notion of web APIs was introduced to indicate those APIs operating over the web. Web APIs are used to provide developers with the building blocks needed to create web-based

software applications. As we are particularly interested in web APIs, in the remainder of this document, unless otherwise specified, the term 'API' will be used to refer to web APIs.

1.2.2 Web APIs and Web Services

There are various definitions of a Web Service. The W3C defines a Web Service as 'a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL— Web Service Description Language). Other systems interact with the web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using Hypertext Transfer Protocol (HTTP) with an XML serialisation in conjunction with other web-related standards' (W3C, 2004). This definition links the concept of a Web Service to a set of specific technologies (SOAP; WSDL; and XML — Extensible Markup Language). Others provide more generic definitions, e.g. (IBM, 2014a) states that a 'Web Service is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location'. (Papazoglou and Georgakopoulos, 2003) define a Web Service as 'a specific kind of service that is identified by a universal resource identifier (URI), whose service description and transport utilise open Internet standards'.

These definitions extend that given by W3C by essentially defining a Web Service as a service that is offered over the web, irrespective of the usage of specific protocols and message formats. Similarly, the OASIS (Advancing Open Standards for the Information Society) reference model for Service Oriented Architecture (OASIS, 2006) defines a Web Service as 'a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description'.

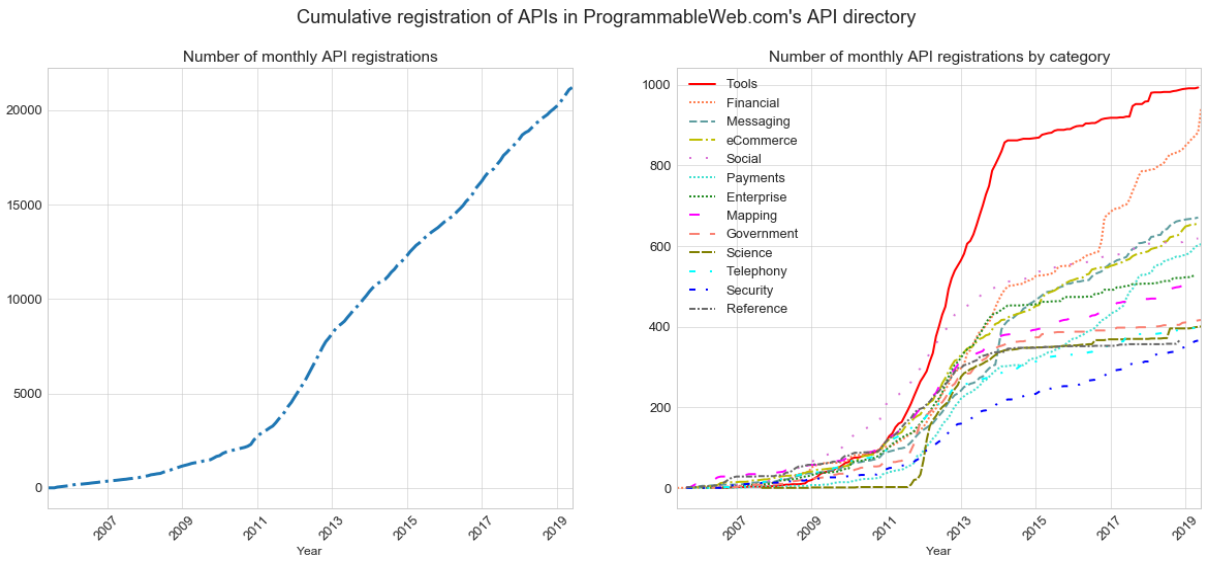
While the generic definitions reported above generalise the restrictive and technology-driven definition made by W3C, they do not clarify the difference between a **service interface** and a **programming interface**: the former is provided by a Web Service, the latter is a distinct characteristic of an API. In this report, however, we consider this difference relevant, since it affects the design of APIs, their implementation and their potential use. Web Service interfaces are designed to offer access to 'high-level' functionalities for end-users, either humans or machines. On the other hand, APIs are designed to provide even 'low-level' functionalities ([2]) as building blocks that can be used and combined by software developers to deliver a higher-level service. Thus, Web Services and APIs differ at the design level but not at the technological level.

1.2.2.1 Trends in the adoption of web APIs

The evidence on the adoption of web APIs comes from ProgrammableWeb.com, which is the primary community resource for amateurs and professionals in the industry. This resource gathers public API end points in a comprehensive directory with information that is self-reported by developers. Figure 1 shows the number of web API records that have been registered since 2005. As of the first quarter of 2019, the ProgrammableWeb directory listed 21 202 records of which 417 had been categorised as 'Government' (primary keyword).

([2]) 'Low level' means that a developer can use such functionalities to build applications, while those same functionalities are not useful for end-users. For a simple example, consider an API providing mapping from a location name to its coordinates; this is not really useful for an end-user, but it is useful for a developer, who can use the API to display the location on a map.

Figure 1. Adoption of web APIs



Left panel: cumulative count of the number of web APIs reported. Right panel: cumulative count of the number of APIs by category.

Source: ProgrammableWeb.com (accessed June 2019 and used with authorisation); own elaboration.

An indication of the purpose of the API is contained in the information about its category, as listed in the directory. Table 1 lists the most frequent categories ([3]) associated with the API records registered. Among the top categories are the financial and e-commerce categories, as well as the payments and enterprise categories. The right panel of Figure 1 also shows that the trends in the number of records of APIs registered in the payments and financial categories increased after the publication of the revised Payment Services Directive (PSD2) (European Union, 2015a), which might have influenced the development of this trend. Finally, we highlight that the government category ranks among the most frequent categories of records.

Table 1. Most common categories of registered web APIs

Rank	First category	Number	Rank	First category	Number
1	Tools	993	**11**	Telephony	398
2	Financial	944	**12**	Security	366
3	Messaging	671	**13**	Reference	366
4	e-commerce	657	**14**	Search	346
5	Social	619	**15**	Email	346
6	Payments	605	**16**	Video	340
7	Enterprise	528	**17**	Travel	321
8	Mapping	510	**18**	Education	311
9	Government	417	**19**	Sports	303
10	Science	401	**20**	Transportation	292

Source: JRC, own elaboration.

1.2.3 Remote Procedure Call and Representational State Transfer

APIs can be broadly categorised into the following main types: (i) RPC APIs and (ii) APIs that adhere to the REST architectural style, or RESTful APIs.

The first category is characterised by a set of procedures or methods that the client application can invoke and are executed by the server to fulfil a task, for example a data exchange or a data validation service call.

([3]) For a definition of the ProgrammableWeb.com API directory data model, see https://www.programmableweb.com/news/programmablewebs-new-api-directory-data-model-explained/analysis/2016/07/08

RPC APIs essentially operate by replacing in-memory object messaging with cross-network object messaging in object-oriented applications (Feng et al., 2009). In a nutshell, this could be exemplified by considering using a code library not in the local environment but over a network, thus sending/receiving messages to/from the code library through the network instead of through the local memory.

RESTful APIs are based on the REST architectural style introduced by (Fielding, 2000). The REST architectural style is a hybrid style derived from several of the network-based architectural styles described in (Fielding, 2000) and combined with additional constraints that define a uniform connector interface. 'The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture. By examining the impact of each constraint as it is added to the evolving style, we can identify the properties induced by the Web's constraints. Additional constraints can then be applied to form a new architectural style that better reflects the desired properties of a modern web architecture. REST defines a set of constraints which restrict the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to REST' (Fielding, 2000).

In essence, here the term 'constraints' refers to the set of characteristics that define the REST architectural style. The constraints defined by REST are outlined below.

- **Client-Server:** a client, in need of a functionality to be performed, sends a request to a server that is capable of providing the functionality.
- **Stateless interaction:** in the interaction between clients and servers, the former do not maintain the resource state and the latter do not maintain the state of the client application.
- **Uniform interface:** client and server components interact by means of a uniform interface that decouples implementations from the services provided. To obtain a uniform interface, the architectural constraints listed below are needed to guide the behaviour of components.
    - **Resource identification:** a resource identifier, generally a URI, is used to identify the particular resource involved in an interaction between components. Examples of resources include a web page, an image or a document.
    - **Self-descriptive messages:** a message sent by a client application to a server contains all information required for its processing.
    - **Manipulation of resources through representations:** resource representations provide clients with all information required to modify the resource.
    - **Hypermedia as the engine of application state (HATEOAS):** in addition to resource representations, server responses also provide the operations that can be performed on such resources, as well as the end points that provide them.
- **Cache:** if a response to a client is cacheable, then the client, or a mediator between the client and the server, is given the right to reuse that response data for later, equivalent requests.
- **Layered system:** this is a hierarchically organised system, where each layer provides functionalities to the layer above it and utilises functionalities of the layer below it.
- **Code on demand:** a client can send a request to a server requesting code needed to process the resource representations; the server provides the code and the client executes it locally. Code on demand is an optional constraint for REST.

These defining properties are the subject of ongoing work by international bodies and consortia. The recent request for comments (RFC) on the Hypertext Transfer Protocol (from RFC 7230 to RFC 7240) and other ongoing work at (IETF, 2019) aim to reorganise existing specifications into a more comprehensive set of documents.

Both RPC and REST require the same understanding of the data model, format and the encoding of messages that are exchanged between the client and the server. In other words, when a message is exchanged, both the client and the server must be able to read it (data format and encoding) and 'understand' its content (data model ([4])). However, the two architectural styles differ in several aspects such as scalability and performance. From an interoperability point of view, the main difference between RPC and REST lies in the degree of client-

([4]) Please note that, in the rest of this section, the term 'data model' is used to refer to both the data format and content encoding.

server coupling, with coupling being tighter for RPC and the REST architectural style allowing looser client-server integration. The degree of coupling has implications on how much a client and a server can evolve independently over long periods but remain interoperable.

In an RPC architecture, the client and the server must share not only the data model, but also knowledge about the set of procedures that can be invoked, their end points and their semantic content. Any unilateral alteration by the server operator to any of these three constraints will adversely affect the client, possibly breaking interoperability. This constraint results in a requirement for tight client-server coupling to preserve interoperability (see also the example in Box 1).

Box 1. Example — RPC style

As an example, consider an API for retrieving posts from a blog. A typical call invoked by the client would have the following form: *`http://<HOST>/posts/?readPost={postid}`*

Where:

— *`http://`* identifies the network protocol;

— *`<HOST>`* indicates the URL of a networked host that communicates with the client software over the network protocol;

— *`/posts/`* is the path on the host, provided by a server software;

— ?*`readPost={postid}`* is an example of a possible concatenation key-value pair.

The API documentation informs the developer that the functionality to retrieve identifiers of available posts is *`<HOST>/posts`* and the response is an array of identifiers; the content of each post can then be retrieved utilising *`<HOST>/posts?readPost={postid}`*, where *`postid`* is the identifier of the desired post. As a consequence, if the API provider changes the second end point, e.g. to *`<HOST>/posts/public?readPost={postid}`*, then the interoperability between client and server will be broken until the client is updated so that it can use the new end point.

On the other hand, REST defines a uniform interface for component interactions. Resource representations are transferred in formats that can be dynamically selected by the client and/or the type of resource they represent. Since the publication of Fielding's doctoral thesis (Fielding, 2000), the adoption of the REST architectural style has gradually increased in popularity. Nonetheless, HATEOAS, a key constraint proposed by Fielding, has yet to be adopted as a mainstream feature of REST. The HATEOAS constraint improves the discoverability of the API, making it self-discoverable; that is, the client can discover not only resources but also their possible state transitions, which are operations that can be performed on such resources (Fielding, 2008). The HATEOAS constraint thus allows a higher degree of client-server decoupling. For example, when a server modifies available operations or end points, the client can automatically discover such modifications and continue to work regularly. This results in a looser client-server coupling, thereby improving client-server interoperability (see also the example in Box 2).

Box 2. Example — REST style

Considering the example in Box 1, a RESTful API replies to the first request (`<HOST>/posts`) not only with a list of identifiers, but also with the end point to be utilised for retrieving post content (e.g. `<HOST>/posts/{postid}`), as prescribed by the HATEOAS constraint. The API documentation must provide the client developer with proper information on how to recognise this end point. When this is the case, the client software discovers not only identifiers of available posts, but also the end point for post content retrieval. If the API provider changes the second end point by altering its path to `<HOST>/posts/public/{postid}`, the client would not need to be updated because it will automatically use the updated end point, thus maintaining interoperability.

Bringing discovery information into the exchange of data adds a higher degree of complexity to APIs that implement HATEOAS and this translates into various costs for developers. In particular, these costs are borne to comply with HATEOAS constraints both in terms of the data model description and documentation, and in terms of the implementation requirements of clients and servers. At the same time, the HATEOAS constraint provides the API with high levels of flexibility and extensibility, which may potentially be of value to both the API provider and consumers. Hence, due to the trade-off between the complexity of HATEOAS implementation

and its value to API providers and consumers, some API implementations do not comply with the HATEOAS constraint.

Generally, whether REST or RPC is adopted depends on a foreseen specific use case. Usually, REST better fits use cases where the provider aims to share the resources with client applications, allowing them to navigate and modify such resources, or when the service may benefit from the distributed nature of HTTP features — e.g. caching, as explained in (Fielding, 2000). On the other hand, RPC is used to share functionalities with client applications that invoke such functionalities to fulfil some task (Maleshkova et al., 2010).

In some cases, development of the existing standards started before the introduction and adoption of REST on a large scale, and standardisation communities are trying to modernise these standards by developing REST versions, for example the OGC Web Services specifications (OGC, 2019b), in particular the Web Feature Service (WFS). The latest WFS version 'OGC API-Features (ex WFS 3.0)' (OGC, 2018a) is a complete rewrite of previous versions, focusing on a simple RESTful core specified as reusable OpenAPI components with responses in JavaScript Object Notation (JSON) and Hypertext Markup Language (HTML) (OGC, 2019c). As a result, the implementation of OGC API-Features will not be backwards compatible with WFS 2.0 implementation, although a design goal has involved defining it in such a way that the new interface can be mapped to an WFS 2.0 implementation (OGC, 2018b). Based on the work for OGC API — Features, OGC is developing RESTful APIs for other existing standards for different types of geospatial data (coverage, catalogues, etc.). All these new versions are still at the draft stage and OGC has organised a hackathon (OGC, 2019d) to collect feedback from developers (June 2019).

1.2.4 API maturity models

API maturity models can be used to assess the level of compliance with some principles defined by the model itself. This is useful not only for providers (e.g. gap analysis or enhancement scheduling) but also for users to select which specific API to use and, in general, how and when it can be used in the most effective way. In the following sections, we present two different models for assessing these aspects: the Amundsen maturity model and the Richardson maturity model.

1.2.4.1 Amundsen maturity model

A well-known design maturity model is the Amundsen model (Amundsen, 2017). This model defines levels of compliance based on the degree of abstraction of the provided API from the underlying implementation. The compliance levels of the Amundsen model are the following:

- **level 0:** data-centric — the internal implementation model is directly exposed at API level;
- **level 1:** object-centric — the API does not directly expose the internal model, but it provides the means (methods) to manipulate objects of the internal model;
- **level 2:** resource-centric — the API is described as a set of resources that can be consumed by client applications; at this level, resources are independent of internal model objects;
- **level 3:** affordance-centric — the API is described as a set of resources utilising hypermedia representations to provide available actions (operations and links) that can be executed on the described resource.

Levels 0 and 1 are considered internal models because they expose internal implementation structures at the API level. Levels 2 and 3 are considered external models because they separate the external model exposed by the API from the internal model used by the implementation.

1.2.4.2 Richardson maturity model

As presented in the previous section, a fully compliant RESTful API must satisfy all constraints defined by the REST architectural style. The Richardson maturity model (Fowler, 2010) is a well-known model for assessing the compliance of RESTful API implementation ([5]). The model defines four maturity levels of implementation (depicted in Figure 2):

([5]) As the author stresses, 'RMM [The Richardson maturity model], while a good way to think about what the elements of REST, is not a definition of levels of REST itself. Roy Fielding has made it clear that level 3 RMM is a pre-condition of REST. Like many terms in software, REST gets lots of definitions, but since Roy Fielding coined the term, his definition should carry more weight than most.'

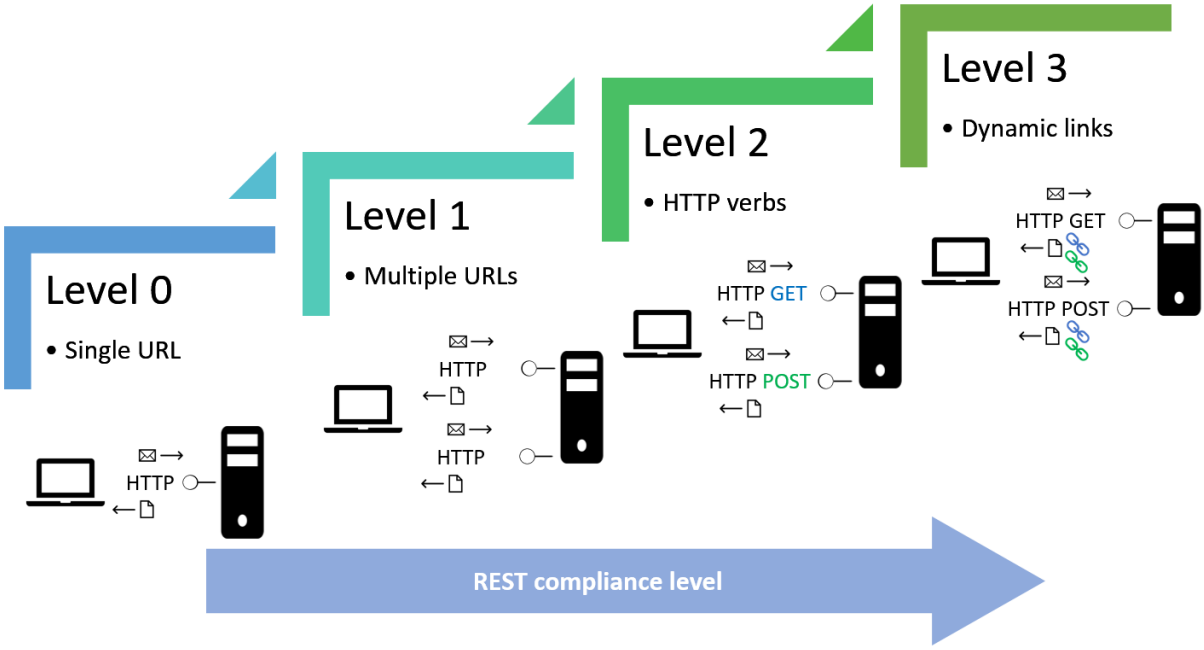
— **level 0:** APIs at this level use HTTP as a transport protocol for remote interactions, generally with a single end point published by the server; essentially these are RPC APIs over the network topology built around the HTTP protocol;

— **level 1:** instead of using a single end point, at this level APIs utilise different end points for the different resources;

— **level 2:** at this level, APIs use HTTP verbs to characterise the requested operation;

— **level 3:** at this level, APIs use HATEOAS, i.e. they use hypermedia to control the transitions of the application; responses from the server provide links for available operations.

Figure 2. Richardson maturity model for assessing RESTful API compliance



Source: JRC, own elaboration.

1.2.5 Microservices

'Microservices address the problem of efficiently building and managing complex software systems. For medium-sized systems, they can deliver cost reduction, quality improvement, agility, and decreased time to market' (Singleton, 2016). No official definition of microservices is available; the National Institute of Standards and Technology (NIST) (Karmel et al., 2016) defines a microservice as 'a basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of well-defined APIs, independent of any vendor, product or technology'. In (Fowler and Lewis, 2014) microservices are defined as 'an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API'. (Newman, 2015) indicates that 'microservices are an approach to distributed systems that promote the use of finely grained services with their own lifecycles, which collaborate together'.

The definitions alluded to above share the idea that microservices provide a way of structuring an application into loosely coupled, independently deployable components that communicate over the web utilising lightweight interfaces. Therefore, microservices deal more with how an application is structured internally than with how it is presented externally to its potential users.

We observe however that the ideas of modularity and loose coupling, as well as the concurrency with which the microservices are built, are also inherent to the so-called Unix philosophy, one of the most successful paradigms in modern software engineering. Microservices can be seen to be the networked, web-enabled analogue of specialised applications running over an operating system, in this case the web or any other network built over a specific protocol.

Microservices only become valuable when they can communicate with other components in a system, i.e. when each of them has an API as its interface. These interfaces also play an essential role in emergent architectural application styles such as the one proposed by using microservices. It is important that, to maintain some fundamental characteristics of the software code (including separation, independence, and modularity), APIs are also loosely coupled. Key design practices required to reach this goal include the hypermedia-driven or HATEOAS implementation (Nadareishvili et al., 2016) described in section 1.2.3.

2 Presentation of the documents

In this section, we analyse the main aspects of the documents we have collected. In particular, we first describe the rationale behind the classification of the documents. Then we present the main features of the documents collected and a shortlist of selected documents. The last part of this section is dedicated to describing European Commission initiatives and standards that we consider related to government APIs.

2.1 Methodology

We have compiled the list of documents from material obtained using different research methods within the APIs4DGov study, namely:

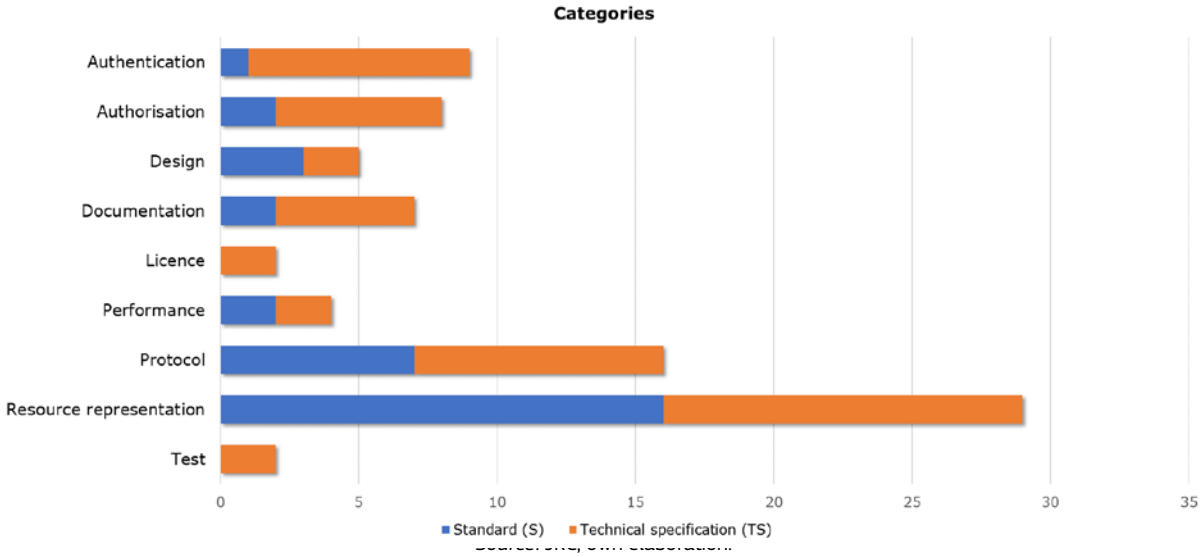
- a desk research activity, which has been integrated and complemented with information from our multiple case study interviews and analysis of government APIs (Williams, 2018);
- our API strategy survey (European Commission, 2018b);
- the 2018 infrastructure for spatial information in Europe (INSPIRE) hackathon (European Commission, 2018c);
- our workshop on government API strategies (European Commission, 2018d);
- a set of interviews with colleagues from other European Commission DGs (e.g. with those involved in the ISA^2 programme) and with relevant domain experts.

The complete list of documents is available in the JRC Data Catalogue (Vaccari and Santoro, 2019). Notice that the list does not include a set of documents which are considered of (i) general purpose for the Web and (ii) consolidated background knowledge of the reader (e.g. HTTP, JSON, XML, URI, SOA, ROA, RDF, etc.). Each document is classified as outlined below.

- **Name:** extended name (with acronym if available).
- **Technical specification or standard:** the documents were separated into two main categories — 'technical specification' or 'standard'. Definitions of these two terms are available in official and technical documents, including the ones at (CEN, 2019; IEC, 2019; ISO, 2019; OGC, 2019e). For the purposes of this report, we use the definitions proposed by the OGC:
  - '**Specification' or 'technical specification' (TS):** 'a document written by a consortium, vendor, or user that specifies a technological area with a well-defined scope, primarily for use by developers as a guide to implementation. A specification is not necessarily a formal standard'.
  - '**Standard' (S):** 'a document that specifies a technological area with a well-defined scope, usually by a formal standardisation body and process'.
- **Category:** each document is classified by its functional specification (resource representation, protocol), security (authentication, authorisation), usability (documentation, design), test, performance and licence. See section 2.2 for a description of each category.
- **Short description:** a short description of the technical specification or standard is given.
- **Link:** the URL for the online document describing the technical specification or standard is given.
- **API type:** information on whether the API is an RPC or REST type is given (both if not specified).
- **Initial release:** the year when the technical specification or standard was first proposed is indicated (where not available, the most probable year, calculated with additional desk research, is indicated).
- **By:** the organisation (i.e. standard body, consortium or vendor) or individual that proposed the standard is indicated.

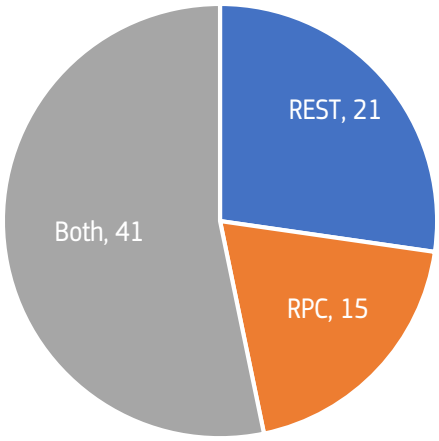
A total of 78 documents were collected, of which 32 can be classified as standards and 41 can be classified as technical specifications. Figure 3 depicts the number and the type of technical specifications and standards by 'Categories' according to the classification illustrated in section 2.2. The resource representation and protocol categories have the largest number of technical specifications or standards, reflecting the high level of available proposals. The licence category has the smallest number of technical specifications and standards, reflecting the fact that, at the moment, licensing is mainly at the data level.

Figure 3. Number of technical specifications and standards per category



Regarding API type, Figure 4 shows that 15 technical specifications and standards are related to RPC and 21 to the REST architectural style. The remainder of the technical specifications and standards can be considered 'general purpose' or neutral with respect to the design style (classified as 'Both'). This distribution of technical specifications and standards reflects the fact that both RPC and REST are widely adopted and that the choice of which type to use is likely to be based on the specific use case to be implemented.

Figure 4. Distribution of API types



Source: JRC, own elaboration.

2.2 Shortlist of technical specifications and standards

This section contains a detailed description of selected technical specifications and standards listed by (Vaccari and Santoro, 2019) and highlights their main characteristics. The shortlisted technical specifications and standards were selected based on their utilisation, maintenance and stability, aiming to provide a snapshot of the current landscape of technical specifications and standards. The shortlist will give the reader basic information about a selected number of technical specifications and standards that have been found to support the study and/or to be of particular (real or potential) importance for administrations engaging in the use of APIs. To assess the relevance of the technical specifications and standards, the following criteria are considered:

— **utilisation:** the estimated ([6]) number of implementations (based on searches in different repositories) of the technical specification or standard available, in terms of providers and/or consumers;

— **maintenance:** whether the technical specification or standard has been maintained (and updated, when needed) or discontinued (non-active);

— **stability:** whether the technical specification or standard is still in a draft version or has stable versions.

Non-active technical specifications and standards are reported in (Vaccari and Santoro, 2019) for completeness, but are not considered for a detailed description. It should be noted that, in addition, draft technical specifications and standards have not been considered for detailed description unless the last available draft was provided after January 2018.

2.2.1 Functional specification

Functional specification is the one of the most relevant aspects to consider in the classification of an API. This entails defining what functionalities the API provides and how such functionalities are provided, that is, defining the resource representation and the interface/protocol used. **Resource representation** is used to categorise all technical specifications and standards dealing with data representation (including data formats, vocabularies, encodings/serialisation). The **communication protocol** ('protocol' for short) category is used to refer to technical specifications and standards describing rules, syntax, semantics, synchronisation of communication and possible error recovery methods, including extensions of consolidated ones (e.g. extensions of HTTP). For each of the categories, we have selected a number of relevant documents from the list in (Vaccari and Santoro, 2019).

2.2.1.1 Resource representation

For the REST architectural style, resource representation is central. Any information that can be named could be a resource: a document or image, a weather forecast service, a collection of other resources, etc. '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' (Fielding, 2000). In (Fielding, 2000), for example, the author's 'preferred version' of an academic paper is a mapping whose value changes over time, whereas a mapping to 'the paper published in the proceedings of conference X' is static; these are two distinct resources, even if they both map to the same value at some point in time. The components (e.g. clients and servers) perform actions on resources by using representations of them. A representation captures the current or intended state of a resource and can be expressed in any message format supported by any two interacting components (e.g. XML, JSON). A fully compliant RESTful API utilises resource representation to not only provide resource descriptions, but also to determine which operations can be performed on the resources as well as the end points that undertake these operations. As noted above, this is known as the REST constraint HATEOAS. The correct implementation of HATEOAS will let a provider implement the 'level 3' of the Richardson maturity model.

The HATEOAS constraint can be specified in a number of different ways and by using a single technical specification or standard or a combination. In the following, we first define some relevant examples of hypermedia specification methods, how to specify media types and the semantics of the links, and, finally, some examples of how to define the semantics of data and links through well-recognised vocabularies.

2.2.1.1.1 Hypermedia specification

For REST, a network-based application can be seen as a state machine (Fielding, 2000). The current state of a server is represented by the hypermedia that is transferred to the client, together with a set of representation metadata (content-type, content-encoding, etc.) that determine the representation. Possible state transitions are represented by the links provided in the hypermedia, which the client can activate or not. In this way, a RESTful API can expose its content in a dynamic way, leading the client through its application states so that it can reach its goal (Liskin et al., 2011). Conceptually, this is equivalent to a user clicking available links on a web page to find the desired document.

([6]) It is not always possible to obtain precise numbers about the utilisation of a specific technical specification or standard. Depending on the specific type of technical specification or standard, utilisation was estimated by searching different repositories, including Google Scholar, scientific literature repositories (Scopus, Web of Knowledge) and grey literature such as development forums (e.g. StackOverflow) and API repositories.

Hypermedia is the key to enabling fully compliant RESTful APIs. Several hypermedia type specifications exist. The most widely known and used specifications are described in the remainder of this section, with a focus on their main characteristics.

— **Hypertext Application Language (HAL):** HAL is a very lightweight specification that provides a set of conventions for expressing hyperlinks in either JSON (ECMA International, 2017) or XML. It is possible to specify resources and links to related resources. HAL specification does not define methods to add or describe actions and operations that can be performed on a resource.

— **JSON for Linked Data (JSON-LD):** JSON-LD is an extension of JSON. Its primary goal is to enable the serialisation of linked data in JSON and has been especially developed for expressing information using structured data (or linked data) vocabularies, such as schema.org. The main concept introduced by JSON-LD is the so-called context. A context in JSON-LD allows two applications to use shortcut terms to communicate with one another more efficiently and without loss in accuracy. Essentially, a context maps terms used in the JSON attributes from a vocabulary (or an ontology) to internationalised resource identifiers (IRIs) ([7]) (IETF, 2005) ([8]). Some useful examples on how to use JSON-LD can be found at (W3C Community Group, 2019). In (Lanthaler and Gütl, 2012), the authors explain the use of JSON-LD in REST. The main advantage of this standard is its full compatibility with JSON, significantly simplifying the transition from JSON-based existing APIs to JSON-LD. With JSON-LD, it is not possible to specify/describe actions that can be performed on a resource; however, JSON-LD can be combined with Hydra Core Vocabulary to provide such a feature.

— **JSON:API:** the JSON:API specification is designed to minimise both the number of requests and the amount of data transmitted in client-server communication. JSON:API's base concept is a resource object. Each resource object has a data field containing the representation of the resource. Apart from the data field, resource objects can have relationships and links fields. In JSON:API, the relationships field provides references to other resource objects related to the current one, while the links field provides URIs relevant for current response (self-pagination, etc.). The wide availability of implementations for different programming languages and frameworks makes this specification widely used. As in the case of HAL, the JSON:API specification, however, also lacks a definition for the available actions that can be performed on resources.

— **SIREN:** SIREN is a hypermedia specification for the representation of entities, offering structures to communicate information about entities, actions for executing state transitions and links for client navigation. In SIREN, an entity represents a resource and is characterised by three elements, namely a class that defines the nature of an entity's content, a set of properties (key-value pairs describing the state of the resource) and a set of links. In SIREN, it is also possible to specify actions, i.e. the operations that can be executed on an entity or resource. Such an action can be defined by providing, for example, the URL of an end point, the HTTP method to be used and control data (parameters).

2.2.1.1.2 Repositories of media and link relation types

When a client and a server exchange representation of a resource, they need to have a common understanding of the format that is used to encode the representation (JSON, XML, etc.). The Internet Assigned Numbers Authority (IANA) is in charge of maintaining a repository of registered formats (media types) for this purpose. IANA also maintains a repository of relation types that specify the semantics of the relationship between the source and the target of a link.

— **IANA media types:** media types (formerly known as MIME types) are defined in RFC 2046 (Borenstein and Freed, 1996) to standardise how to specify the nature of data in the body of a message. IANA maintains a registry of media types available online. For each media type, the registry provides the recognised value to be used in message exchange and a link to its specification. A standardised procedure (Klensin et al., 2013) is available to define and request the registration of new media types.

([7]) IRIs were defined to extend the existing uniform resource identifier (URI) scheme. While URIs are limited to a subset of the ASCII character set, IRIs may contain characters from the Universal Character Set (Unicode/ISO 10646), including Chinese or Japanese kanji, Korean, Cyrillic characters, and so forth.

([8]) See, for example, how the '@context': http://schema.org and the '@type': 'Book' are mapped to the IRI http://schema.org/Book (http://www.linkeddatatools.com/introduction-json-ld).

— **IANA link relation types**: a link relation type identifies the semantics of a link ([9]), i.e. the relationship between the source and the target of the link. In HATEOAS, the use of shared and standardised link relation types is key, since the links represent available state transitions. Client applications can use the provided links and, based on their understanding of the link relation type, follow the one needed to implement their business logic ([10]). In addition, semantic web technologies rely on the use of shared and standardised link relation types, which enable the automatic processing of machine-readable statements. IANA maintains a registry of link relation types, which can be used to describe how different resources are linked.

2.2.1.1.3 Vocabularies

Vocabularies are key to enabling interoperability between clients and servers. Their use allows the two interacting components to have a common understanding of the resources described. An API might use its own vocabulary or an existing one. This second option increases the interoperability level, potentially allowing any client that utilises the same vocabulary to use the API.

Several vocabularies exist, covering different application domains and themes. Some examples are given in Box 3.

Box 3. Examples of vocabularies

The **ISA core vocabularies** are simplified, reusable and extensible data models that capture the fundamental characteristics of an entity in a context-neutral way. This action is supported by the European Commission's ISA[2] programme for supporting the modernisation of public administrations in Europe through the development of e-government solutions. So far, the following core vocabularies are available: (i) Core Person Vocabulary (captures the fundamental characteristics of a person such as name, gender and date of birth), (ii) Core Location Vocabulary (captures the fundamental characteristics of a location, represented as an address, a geographical name or a geometry), (iii) Core Business Vocabulary (captures the fundamental characteristics of a legal entity, e.g. its identifier and activities), (iv) Core Public Service Vocabulary (captures the fundamental characteristics of a service offered by a public administration such as the title and description of the service), (v) Core Criterion and Evidence Vocabulary (describes the principles and means that a private entity must fulfil to qualify to perform public services), and (vi) Core Public Organisation Vocabulary (captures the fundamental characteristics of public organisations in the European Union, e.g. the contact point and address).

Another widely used repository is **schema.org**; it collects data models and vocabularies from different thematic communities and provides a collection of shared vocabularies that can be used to describe resources and links between resources. The schema.org vocabulary can be used with many different encodings, including RDFa (Resource Description Framework in Attributes), Microdata and JSON-LD. These vocabularies cover entities, relationships between entities and actions, and can easily be extended through a well-documented extension model.

Hydra Core Vocabulary is the outcome of recent efforts by the W3C to provide a vocabulary that enables a server to advertise valid state transitions to a client, i.e. implement the relevant HATEOAS REST constraint. The Hydra Core Vocabulary defines how to use JSON-LD for specifying actions that can be executed on a resource. To this end, the notion of **operations** is introduced in the JSON-LD representation of a resource. An operation provides the information necessary for a client to construct valid HTTP requests to manipulate the server's resource state, including the HTTP method and control data. Moreover, an operation can also specify the possible returned resources and status codes from the server. The Hydra Core Vocabulary is currently in draft (last unofficial version released in March 2019).

2.2.1.2 Communication protocol

Communication protocols are formal descriptions of digital message formats and rules. They are required to exchange messages in or between computing systems and are required in telecommunications. In our case, we focus on specific protocols, as outlined below.

([9]) Identified, in HTML, by the tags 'a' (anchor) and link.

([10]) For example, 'next' is a relation type used to indicate that the link's context is a part of a series, and that the next in the series is the link target.

2.2.1.2.1 GraphQL

GraphQL is a query language designed to build client applications by providing an intuitive and flexible syntax and system for describing their data requirements and interactions (GraphQL, 2018). It supports reading, writing (via mutations ([11])) and subscribing to changes of data (real-time updates). GraphQL is of interest because its specification defines an application-level protocol for client-server communication, in particular the syntax and semantics of messages. These are transferred using HTTP as a message transfer protocol.

The basic idea behind GraphQL is to increase the performance of client-server communication. By allowing the client to specify, in one single data structure, both the query and the response format requested to the server, GraphQL prevents the growth of message exchange between client and server, as well as the message size. This flexibility gives a lot of power to clients but makes request processing more computationally intensive for servers. In addition, GraphQL provides a runtime to execute GraphQL queries with existing data on the server and a large set of client-side and server-side libraries in different programming languages.

Recently, IBM cloud researchers released version 1.0.0 of OpenAPI-to-GraphQL, a library for autogenerating GraphQL wrappers for existing REST(-like) APIs (Cha, 2019). The library is based on an IBM research paper that describes the challenges of building OpenAPI-to-GraphQL. The authors evaluated OpenAPI-to-GraphQL against 959 publicly available OpenAPI specifications (OASs), provided by APIs.guru, and successfully created GraphQL interfaces for 89.5 % of them (Wittern et al., 2018).

In (Taelman et al., 2018), the authors observe that the most popular (web) application frameworks, such as React and Angular, have limited support for querying the web of linked data, and instead GraphQL is tightly integrated into these frameworks. To lower the barrier for developers with regard to linked data consumption, the authors introduce GraphQL-LD, an approach that consists of a method for transforming GraphQL queries coupled with a JSON-LD context to SPARQL (SPARQL Protocol and RDF Query Language), and a method for converting SPARQL results to the GraphQL query-compatible response. HyperGraphQL is also a GraphQL interface for querying and serving linked data on the web. It is designed to support federated querying and exposing data from multiple linked data services using GraphQL query language and schemas. The core of the response is a JSON-LD object, which extends the standard JSON with the JSON-LD context, enabling semantic disambiguation of the contained data (Semantic Integration, 2019).

2.2.1.2.2 gRPC

gRPC was initially developed by Google and is now maintained by the Cloud Native Computing Foundation as an open-source project. gRPC is designed for both the high-performance and high-productivity design of distributed applications. Like many RPC systems, gRPC is based around the idea of defining a service, and specifying the methods that can be called remotely with their parameters and return types. Developers can specify the methods, parameters and return types that are invoked by client applications. The API is implemented by a gRPC server that executes client calls. gRPC uses HTTP as the transport protocol and protocol buffers for the Interface Definition Language (IDL) and the underlying message interchange format. Additional features such as authentication, bidirectional streaming ([12]) and flow are also supported. gRPC provides a set of libraries in different programming languages for its use in different environments.

2.2.1.2.3 SPARQL

SPARQL is a widely used query language for the Resource Description Framework (RDF). SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF format or viewed as RDF format via middleware (W3C, 2013a). The SPARQL query language specification has been extended by the SPARQL update (W3C, 2013b), which defines a standard language for specifying and executing updates to RDF graphs in a graph store. The SPARQL protocol essentially uses RPC for interacting with SPARQL processors.

The SPARQL 1.1 protocol (W3C, 2013c) describes how to convey SPARQL queries and updates from clients to SPARQL processors. The SPARQL protocol uses HTTP as the transfer protocol, to allow client and server to exchange SPARQL query and results. The specification describes two HTTP operations: a query operation for performing SPARQL queries and an update operation for performing SPARQL update requests. SPARQL protocol clients send HTTP requests to SPARQL protocol servers, which handle the request and send HTTP responses back to the originating client. The W3C also defined the SPARQL 1.1 graph store HTTP protocol

([11]) A GraphQL term used to indicate a way to modify server-side data.
([12]) Both client and servers send a sequence of messages using a read-write stream (https://www.grpc.io/docs/guides/concepts/).

(W3C, 2013d). This document applies some of REST-style principles to specify how to use HTTP operations for the purpose of managing a collection of RDF graphs.

2.2.1.2.4 WebSocket

The WebSocket protocol allows the creation of full-duplex, bidirectional connections between a client and a server over the web. The goal of this technology is to provide a mechanism for client applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (Fette and Melnikov, 2011). The protocol allows a user to create persistent and lower-latency connections that can be handled by both client and server. An interesting feature of the WebSocket protocol is that it allows servers to push information (e.g. events) to clients, since the established communication is bidirectional and persistent. The WebSocket protocol is supported by all major browsers currently available and many software libraries. As the WebSocket protocol — once the connection is established — does not rely on HTTP semantics, auditing should be done entirely at the application level. However, utilising HTTP guarantees that applications utilising the WebSocket protocol can work through proxies and firewalls.

2.2.1.2.5 XML-RPC and JSON-RPC

JSON-RPC and XML-RPC are specifications that define how to use, respectively, JSON and XML for encoding messages that represent an invocation of some method on the server. The messages are then passed to the server over HTTP. The server executes the requested method and returns the result in the HTTP response.

2.2.2 Security

In general, security deals with aspects related to authentication, authorisation and digital signature/encryption. Another important facet of security, in particular for digital government, is eID. **Authentication** is the ability to prove that a user or application is genuinely who that person or what that application claims to be (IBM, 2014b; ENISA, 2019; NIST, 2019). For simplicity, in this report the authentication category also includes electronic identity documents. **Authorisation** protects critical resources in a system by limiting access to only authorised users and their applications (IBM, 2014c). See Box 4 for more information about API security. As well as the data themselves, there are several technologies and standards relevant for data access through APIs in terms of both user authentication and authorisation.

Box 4. API security

Security is a crucial aspect of API development. In fact, APIs provide another channel into an organisation's resources and information and offer direct, machine-to-machine access to resources and information, which makes it less easy to determine whether information is correctly or incorrectly exposed (Department of Internal Affairs - Government of New Zealand, 2016).

In (Wang et al., 2013) a set of apps is analysed to assess their vulnerability. The study considered apps developed with Software Development Kits (SDKs) provided by major online providers for incorporating authentication functionalities. The study determined that, even when developers follow accepted programming procedures, 67 % to 86 % of the apps analysed had security vulnerabilities, mainly due to implicit assumptions that are not readily apparent to app developers, potentially leading to users having their system credentials stolen.

There is no single strategy or technology which can ensure a secure development out of the box. APIs need to be secure by design, depending on the specific use case they are implementing, which requires security to be built in from scratch, and be considered within the context of existing protection mechanisms. In (Department of Internal Affairs - Government of New Zealand, 2016) the authors give a series of useful standards and guidelines to support the implementation of the security layer.

For each of the categories, we have selected a number of relevant documents from the list in (Vaccari and Santoro, 2019).

2.2.2.1 Authentication

2.2.2.1.1 API key

API keys are not defined by any formal standard or technical specification. However, their use is a very common practice in implementations, mainly for authentication purposes. An API key is akin to a password, which the client application attaches to a request message and the server uses to identify the calling entity (e.g. the client application or the end-user). API keys can be used either to control how the API is being used, e.g. to prevent its malicious use or abuse, or to implement a simple authentication and sometimes authorisation mechanism.

In general, API keys are not considered secure. In fact, after providing the key, servers have no control over how securely the key is used; for example, since API keys are usually accessible to clients, it is easy for someone to steal an API key and utilise it. There is no possibility for the provider to verify that the incoming request is making use of a stolen key.

2.2.2.1.2 OpenID Connect

OpenID Connect or simply OpenID provides an authentication layer on top of the OAuth 2.0 specification. In a nutshell, OpenID exploits the OAuth 2.0 delegated authorisation mechanism to provide a federated authentication functionality. The main extensions defined by OpenID are:

- additional scope in the OAuth 2.0 authorisation request: OpenID;
- a new data structure, ID token, encoded as a JSON web token (JWT) (Bradley et al., 2015a);
- a UserInfo end point that client applications can use to request additional information about the end-user.

The OpenID scope is used by client applications requesting to use the OpenID extension for OAuth 2.0 servers.

Information about the authentication of the end-user is provided in the ID token data structure, in the form of claims. The ID token is signed using the JSON Web Signature (JWS) (Bradley et al., 2015b) and optionally encrypted with JSON Web Encryption (JWE) (Hildebrand and Jones, 2015).

The UserInfo end point is defined as an OAuth 2.0 Protected Resource which can be accessed utilising the OAuth 2.0 Access Token obtained during the authentication process.

The OpenID authentication steps are the following:

1. the client application (relying party — RP) sends a request to an OpenID-Connect-compliant server (OpenID provider — OP);
2. the OP authenticates the end-user and obtains authorisation;
3. the OP responds to the RP with an ID token and usually an access token;
4. the RP can send a request with the access token to the OP UserInfo end point;
5. the OP UserInfo end point returns claims about the end-user.

The OpenID specification suite is modular, supporting optional features such as the encryption of identity data, the discovery of OpenID providers and session management. Implementations can comply at different levels:

- **core:** defines the authentication layer on top of OAuth 2.0 and specifies how to use claims for communicating end-user information;
- **dynamic:** clients can dynamically discover information about OPs and register with them;
- **complete:** defines the form post-response mode (i.e. how to use OAuth 2.0 in combination with HTML forms) and session management (login/logout).

OpenID utilises HTTP GET/POST methods for message exchange between clients and servers.

2.2.2.1.3 Security Assertion Markup Language 2.0

Security Assertion Markup Language (SAML) 2.0 is a standard for exchanging authentication and authorisation information between different entities. SAML messages are based on XML, expressed as SAML assertions trusted by all entities involved in the message exchange. SAML defines:

- **assertions:** statements about security information about a subject principal (usually an end-user) between a SAML authority, named an 'identity provider', and a SAML consumer, named a 'service provider';
- **protocols:** these define request/response messages in SAML (e.g. authentication requests);
- **bindings:** the mapping of SAML request-response message exchanges on to standard messaging or communication protocols; SAML defines bindings for SOAP, Reverse SOAP (PAOS), HTTP Redirect, HTTP POST, HTTP Artefacts and SAML URI;
- **profiles:** a profile specifies how assertions, protocols and bindings are combined to provide interoperability; SAML defines a selected set of profiles and provides guidelines to specify new ones.

The SAML standard allows the implementation of federated identity, linking and using the electronic identities a user has across several identity management systems. With federated identity, a client application can authenticate a user without the need to make use of its internal user database. Instead, client applications can use trusted third-party identity management systems for user authentication.

A Common Assessment Method for Standards and Specifications (CAMSS) report is available for SAML (CAMSS Team, 2019a), which classifies SAML as compliant with the EU Regulation on European standardisation (European Union, 2012).

2.2.2.2 Authorisation

2.2.2.2.1 Extensible Access Control Markup Language

Extensible Access Control Markup Language (XACML) is an access control standard. XACML is based on XML (with existing OASIS drafts for JSON profiles) and defines an attribute-based access control system but can also be used to implement role-based access control.

XACML defines base concepts (policy set, policy and rules) and language for expressing an access control policy. The following major actors are part of the authorisation flow defined by XACML:

- **policy administration point (PAP):** the system entity that creates a policy or policy set;
- **policy decision point (PDP):** the system entity that evaluates requests against applicable policy and renders an authorisation decision;
- **policy enforcement point (PEP):** the system entity that performs access control, by making decision requests and enforcing authorisation decisions;
- **policy information point (PIP):** the system entity that acts as a source of attribute values;
- **context handler:** XACML core language is insulated from the application environment by the XACML context defined in XML schema, describing a canonical representation for the inputs and outputs of the PDP; the context handler must convert between the attribute representations in the application environment and the XACML context.

Users request access to the PEP, which (via a context handler) sends a request to the PDP for the attributes. The PDP applies the policy set obtained from the PAP and, if necessary, the attributes from the PIP. The calculated response (permit, deny, indeterminate or not applicable) is finally returned by the PDP to the PEP, which implements the necessary application-specific business logic to handle the PDP response.

XACML does not standardise any API for interaction, either internally (PAP, PDP, etc.) or externally (e.g. PEP for requesting access). Domain-specific extensions of XACML were developed to address specific needs; for example, GeoXACML (OGC, 2011) provides support for spatial data types and spatial authorisation decision functions.

2.2.2.2.2 OAuth 2.0

OAuth 2.0 is a standard for authorisation, specifically addressing delegated authorisation, i.e. a system that allows a party that holds some authorisation to delegate a subset of that authorisation to another party, without requiring either party to disclose credentials to the other (IETF, 2019). With delegated authorisation, third-party applications can be granted authorisation to access resources they do not own, without the need for the resource owner to share security credentials.

The key roles defined by the OAuth standard are:

- **resource owner:** the end-user owning the resource;
- **client:** the client application requesting access to the resource;
- **resource server:** the server hosting the resource;
- **authorisation server:** the server issuing authorisations.

The OAuth standard defines different processes (flows) to achieve delegated authorisation. These flows essentially differ in how the client application obtains the required authorisation grant. The authorisation grant is a credential representing the resource owner's authorisation to access the protected resource. OAuth defines four authorisation grant types — (i) authorisation code, (ii) implicit, (iii) resource-owner password credentials and (iv) client credentials — and, depending on which authorisation grant is used, a corresponding OAuth flow is defined and executed.

After obtaining the authorisation grant, the client uses it to request an access token for the authorisation server. The access token is the credential that the client can use to access the resource on the resource server. Although OAuth 2.0 is specifically designed to operate over HTTP, no specific APIs are defined in the standard.

A CAMSS assessment report is available for OAuth 2.0 (CAMSS Team, 2018), which classifies OAuth 2.0 as compliant with the EU Regulation on European standardisation (European Union, 2012).

2.2.3 Usability

Usability deals with the ease of use of the API by third-party applications (and their developers). Particularly relevant usability concepts for interoperability are documentation and design. A well-documented and -designed API is essential for its integration into web applications and mashups. **Documentation** (or definition) is a technical content deliverable, containing instructions on how to effectively use and integrate with an API (Swagger.io, 2019a). This category is used to classify all documents specifying how to provide either human- or machine-readable documentation. The **design** category is used to refer to documents providing guidance, principles and best practices for software development. For each of the categories, we have selected the two most relevant documents.

2.2.3.1 Documentation

2.2.3.1.1 OpenAPI specification

The OAS, formerly known as the Swagger 2.0 specification, is a standard to describe RESTful APIs. This standard is designed to allow both humans and machines to understand the capabilities of an API. OAS documents can be encoded in either YAML Ain't Markup Language (YAML) or JSON formats. An OAS definition can be used by documentation-generation tools to display the API, code-generation tools to generate servers and clients, testing tools and many other use cases. The OAS does not mandate a specific development process (e.g. design-first or code-first development).

The OAS is a community-driven open specification within the OpenAPI Initiative, a Linux Foundation Collaborative Project (OAI, 2019). The standard is currently in version 3.0.2 (June 2019). It is worth noting that version 3 introduced the important feature of describing links in response bodies. This represents a step towards fully supporting REST. Currently, the OAS does not provide any explicit support for documenting HATEOAS, although there are ongoing discussions in the community about this topic (OAS Community, 2019).

A CAMSS assessment report is available for OAS 3.0 (CAMSS Team, 2019b), which classifies OAS 3.0 as compliant with the EU Regulation on European standardisation (European Union, 2012).

2.2.3.1.2 API Blueprint and RESTful API Modeling Language

API Blueprint and RESTful API Modeling Language (RAML) are two other popular specifications for description and documentation. API Blueprint syntax is based on Markdown Syntax for Object Notation (MSON), while RAML syntax supports both YAML and JSON. None of the two specifications provides explicit support for HATEOAS.

Both API Blueprint and RAML have joined the OAS community in the last 2 years, although they still maintain their own specifications.

2.2.3.2 Design

2.2.3.2.1 European interoperability framework

The European Interoperability Framework (EIF) is a commonly agreed approach for the delivery of European public services in an interoperable manner (European Commission, 2017a). Even if not specific for APIs, the EIF provides public administrations with a set of recommendations to improve governance of their interoperability activities, establish cross-organisational relationships and streamline processes supporting end-to-end digital services, and provides the means to ensure that existing and new legislation do not compromise interoperability efforts. In particular, according to the underlying principle of transparency, the recommendation is to 'ensure internal visibility and provide external interfaces for European public services'.

The EIF promotes the idea of 'interoperability by design', meaning that interoperability aspects should be taken into account during the design phase, in accordance with the proposed EIF model.

2.2.3.2.2 Future Internet Ware

FIWARE (Future Internet Ware) is a curated framework of open-source platform components aimed at accelerating the development of smart solutions. It defines a universal set of standards for context data management that facilitate the development of smart solutions for different domains such as smart cities, smart industry, smart agrifood and smart energy. For any smart solution, there is a need to gather and manage context information, process that information and inform external actors, enabling them to actuate and therefore alter or enrich the current context.

The FIWARE Context Broker (see below) is the core component of any 'powered by FIWARE' platform. It enables the system to perform updates and access the current state of context. The FIWARE Context Broker in turn is surrounded by a suite of additional platform components, which may supply context data (from diverse sources such as a customer relationship management system, social networks, mobile apps or IoT sensors), supporting the processing, analysis and visualisation of data or providing support for data access control, publication or monetisation (FIWARE Foundation, 2019).

2.2.3.2.3 Open Data Protocol

The Open Data Protocol (OData) is a standard for building RESTful APIs. It defines a set of best practices for building and consuming them (OData, 2019). OData also provides guidance on tracking changes, defining functions/actions for reusable procedures and sending asynchronous/batch requests, etc. The design principles of OData are:

- variety: support mechanisms that work on a variety of data stores;
- extensibility: APIs should be able to support extended functionality without breaking clients unaware of those extensions;
- REST principles;
- building incrementally: a basic, compliant API should be easy to build, with additional work necessary only to support additional capabilities;
- simplicity: address the common cases and provide extensibility where necessary.

Moreover, the OData standard also defines the description/documentation of an API; it supports the description of data models, and the editing and querying of data according to those models by specifying:

- metadata: a machine-readable description of the data model exposed by a particular data provider;
- data: sets of data entities and the relationships between them;

— querying: requesting that the service perform a set of filtering and other transformation steps to its data, then return the results;

— editing: creating, updating and deleting data;

— operations: invoking custom logic;

— vocabularies: attaching custom semantics.

2.2.4 Test

This category is used to refer to the documents and tools used for API testing by users (i.e. application developers). The documents and tools we have selected are listed below.

— **Postman collections:** widely used tools for testing (and development), which allow users to define an executable version of documentation via its Postman collections (Postman, 2019). These documents can be shared with client developers to provide them with a predefined entry point for API testing.

— **Swagger:** Swagger (Swagger.io, 2019b) is an open-source software framework consisting of several tools for designing, building, documenting and consuming APIs. The most widely known is Swagger UI, a tool that, given an OpenAPI document, produces an HTML page for testing purposes. Other tools provided by Swagger include those that provide support for automated documentation, code generation and test-case generation.

2.2.5 Performance

This category is used to classify documents, describing either methodologies to assess performances or service-level agreements (SLAs). In this category, the technical specifications and standards related to scalability and reliability are also considered. **Scalability** is the capability of a system to handle a growing amount of work. A formal definition is given by ISO and the International Electrotechnical Commission (IEC) (ISO and IEC, 2016): in the case of the underlying infrastructure, such as cloud services, 'Rapid elasticity and scalability is "A characteristic of cloud computing where physical or virtual resources can be rapidly and elastically adjusted, in some cases automatically, to quickly increase or decrease resources"'. **Reliability** is the assurance that the system is behaving and responding as intended. **Availability** is the property of being accessible and usable upon demand by an authorised entity. Performances are usually measured by ad hoc solutions based on private companies' solutions (see, for example, (APImetrics, 2019), used by the US government) or developed directly by governments (see, for example, (UK Government, 2019) or, again, set up (some time ago) by a restricted group of private companies (Apdex Alliance, 2007). A more general standards document, which can be used as reference for the cloud, is ISO/IEC 19086-1:2016 (ISO and IEC, 2016). It provides an overview, foundational concepts and definitions for the cloud SLA framework. ISO/IEC 19086 builds on the cloud computing concepts defined in ISO/IEC 17788 (ISO and IEC, 2014a) and ISO/IEC 17789 (ISO and IEC, 2014b). It can be used by any organisation or individual involved in the creation, modification or understanding of a cloud SLA that conforms to ISO/IEC 19086. The cloud SLA should account for the key characteristics of a cloud computing service and needs to facilitate a common understanding between cloud service providers and cloud service customers.

2.2.6 Licence

The different types of APIs (private, restricted or open) will result in different approaches to licensing. For the private and restricted cases, joint contractual service licence agreements should define the terms of use (usage, distribution, modification, etc.) and ensure security measures are agreed upon as well. One example built for private companies that would need to adopt a common template for the definition of such a licence is the one proposed by the Swedish Governmental Agency for Innovation Systems (Swedish Governmental Agency for Innovation Systems, 2019). Normally, licensing considerations for APIs are not a simple operation and require thinking through the different layers of the API stack including the server's code, the data layer, the definition of the interface and the client code. Each of these layers can have specific licence considerations (Lane, 2015). Many options are available to specify the licensing of each of the layers.

A relatively complete collection of these licences and information on how to specify them in a structured way are provided by the Software Package Data Exchange (SPDX) (SPDX Workgroup-Linux Foundation, 2019). SPDX is an open standard for communicating software bill of material information (including components,

licences, copyrights and security references). The uniqueness of this approach is that it is possible to codify the appropriate licence in each module of the software code. It also reduces redundant work by providing a common format for companies and communities to share important data about software licences, copyrights and security references, thereby streamlining and improving compliance. Linked to SPDX, as from June 2019, JoinUp proposes a new solution: the JoinUp Licensing Assistant (JLA). The JLA is a tool that allows everyone to compare and select licences based on their content (European Commission, 2019b). Creative Commons (Creative Commons, 2019a) also proposes a web tool that allows the user to select the appropriate Creative Commons licence. The user can specify many licence features including sharing adaptation of the work and allowing commercial use of the work. The system returns the licence that best fits the user's needs (Creative Commons, 2019b).

Regarding open-source licences in particular, a community of GitHub developers has proposed a guide for understanding the legal implications of open source and explaining which open-source licence is appropriate for a specific project (GitHub, 2019a). In addition, GitHub supports developers in choosing an open-source licence for their source code (GitHub, 2019b). The website does not provide a comprehensive directory of open-source licences but rather lets the user choose from a set of the most commonly used software licences and has an appendix that allows the list of licences it proposes to be checked. The Open Source Initiative proposes a set of frequently asked questions (FAQs) that help the user to choose the right licence from a set of open-source software licences. Questions are related to choosing the best open-source licence and how to apply the source licence to the released software. It also gives some advice on what to do if the user violates a copyleft licence and on the meaning of 'contributor agreements' (Open Source Initiative, 2019). The Free Software Foundation recommends the steps and illustrates differences when choosing licences for a software developer work. They recommend choosing different licences for different projects, depending mostly on the software's purpose and in particular for small programs, libraries and server software (Free Software Foundation, 2018).

2.3 European Commission initiatives and related standards

The aim of this section is to provide some insight into current regulations and initiatives of the European Commission that relate to API technical specifications and standards and that, even if they do not specifically target APIs, have emerged thanks to the meetings and discussions with a number of colleagues of the European Commission.

2.3.1 Directive on open data and the reuse of public sector information

The Directive on open data and the reuse of public sector information (European Union, 2019) provides a common legal framework for a European market for government-held data (public sector information). It is built around two key pillars of the internal market: transparency and fair competition. This directive, also known as the Open Data Directive, entered into force on 16 July 2019 and replaces the PSI Directive (Directive 2003/98/EC), which dated from 2003 and was subsequently amended by Directive 2013/37/EU.

The new directive introduces substantive changes to the past legal text. To fully exploit the potential of public sector information for the European economy and society, there should be a focus on the following areas: the provision of real-time access to dynamic data via adequate technical means; increasing the supply of valuable public data for reuse, including from public undertakings, research-performing organisations and research funding organisations; tackling the emergence of new forms of exclusive arrangements; the use of exceptions to the principle of charging the marginal cost; and the relationship between this directive and certain related legal instruments. Even if the directive does not specify any particular API standard or technical specification, APIs are explicitly mentioned for the two former types of datasets, namely dynamic and high-value datasets.

The publication of dynamic data (including environmental, traffic, satellite, meteorological and sensor-generated data) is of particular importance, as their economic value depends on the immediate availability of the information and regular updates. Dynamic data should therefore be available immediately after collection or, in the case of a manual update, immediately after the modification of the dataset via APIs so as to facilitate the development of internet, mobile and cloud applications based on such data. The setting up and use of an API needs to be based on several principles: availability, stability, maintenance over life cycle, uniformity of use and standards, user-friendliness and security.

To provide for conditions supporting the reuse of documents that are associated with important socioeconomic benefits with a particularly high value for the economy and society, a list of thematic

categories for high-value datasets should be set out. For the purpose of ensuring their maximum impact and to facilitate reuse, the high-value datasets should be made available for reuse with minimal legal restrictions and free of charge. The high-value datasets should also be published via APIs.

2.3.2 Revised Payment Services Directive

The first PSD (European Union, 2007a) regulated the information requirements, the rights and the obligations of payment service users, and the requirements of payment service providers (PSPs) for entering the market.

In 2015, a revised version of the PSD (PSD2) was published (European Union, 2015a). This revised directive introduced several changes; for the scope of this report, the most relevant change is the introduction of third-party actors in the payment service market. PSD2 defines the actors listed below (European Payment Council, 2019):

- Third-party payment service providers (TPPs): a TPP is a payment institution that does not hold payment accounts for its customers and provides payment initiation and/or account information services. It can act as:
  - an account information service provider (AISP): for the aggregation of online information for multiple payment accounts to offer a global view of the customer's daily finances, in a single place, to help them better manage their money;
  - a payment initiation service provider (PISP): for the facilitation of online banking for making payments.
- Account servicing payment service providers (ASPSPs): an ASPSP is for the provision and maintenance of a customer's payment account. Credit institutions, payment institutions and electronic money institutions can be ASPSPs, but also AISPs and PISPs.

At the technical level, PSD2 is supported by regulatory technical standards (RTS) that include an API definition to help enable interoperability among banks and third parties. These RTS were developed by the European Banking Authority (EBA) in close cooperation with the European Central Bank (ECB). The RTS specify the requirements of strong customer authentication (SCA), the exemptions from the application of SCA, and the requirements for common and secure open standards of communication between account ASPSPs, PISPs, AISPs, payers, payees and other PSPs (EBA, 2017). The final version of the RTS was approved by the European Parliament and Council in March 2018 and entered into force in September 2019.

The RTS define how the customer's account information is shared between the ASPSP, the PISP and the AISP by requiring that: (i) customers have to give their explicit consent for the TPP to share their payment account data or to initiate a payment transaction, and (ii) ASPSPs provide the TPPs SCA to enable access to the payment account.

The provision of SCA means essentially that ASPSPs must implement, document and publish a dedicated interface that PISPs and AISPs can utilise to retrieve customer information or initiate a payment. Although neither the directive nor the RTS explicitly mention the use of APIs, among financial institutions and FinTech (financial technology) companies active in the sector, they have been proposed as desirable technologies to adopt (PWC, 2016). One of the two possible secure communication channels (provided by the ASPSP to the AISP or PISP) can be provided via a dedicated communication interface. This is concretely translated into the creation of an API.

Currently, there are several standardisation initiatives for PSD2-compliant APIs, including BBVA API Market (BBVA, 2019), UK Open Banking (Open Banking, 2019), the Berlin Group NextGenPSD2 (Berlin Group, 2019), the PolishAPI (PolishAPI, 2019) and STET PSD2 APIs (STET, 2019).

2.3.3 The INSPIRE Directive

The INSPIRE Directive (European Union, 2007b) aims to create a European spatial data infrastructure for the purposes of EU environmental policies and policies or activities that may have an impact on the environment. This European spatial data infrastructure will enable the sharing of environmental spatial information among public sector organisations, facilitate public access to spatial information across Europe and assist with policymaking across boundaries.

INSPIRE is based on the infrastructures for spatial information established and operated by the Member States of the European Union. The directive addresses 34 spatial data themes needed for environmental

applications. It came into force on 15 May 2007 and has been implemented in various stages, with full implementation required by 2021 (European Commission, 2019c).

To ensure that the spatial data infrastructures of the Member States were compatible and usable in a community and transboundary context, the INSPIRE Directive required that common implementing rules (IRs) were adopted in a number of specific areas, including for specific web services (European Commission, 2007a; European Commission, 2007b). A draft mapping, between what INSPIRE requires in terms of download services, has already been done for two API standards, namely the OCG-API Features (Lutz et al., 2019) and SensorThing API (Kotsev et al., 2018). Moreover, the European Commission recently launched a call for tenders, within the ISA^2 programme (European Union, 2015b), to facilitate access to INSPIRE data through standard-based APIs. The call aims to investigate the feasibility, design and implementation of geospatial APIs that leverage on the investment made by EU Member States in the implementation of the INSPIRE Directive (European Commission, 2019d).

2.3.4 Single Digital Gateway

The Single Digital Gateway (SDG) (European Commission, 2018e) is a regulation that aims to eventually allow citizens and businesses to benefit from fully electronic public services in a cross-border manner by the end of 2023 for 21 procedures. This will require some fundamental changes to how information about public services is exchanged and made available publicly. The European coordinator of the SDG has to collect the descriptions of public services from European public administrations in one unique portal; such collection would be automated to prevent problems caused by human error and eliminate the need for manual updates. Member States and the Commission should aim to provide links to a single source of the information required for the gateway to avoid confusion among users as a result of different or fully or partly duplicative sources of the same information. This should not exclude the possibility of providing links to the same information offered by local or regional competent authorities regarding different geographical areas. It should also not prevent some duplication of information where this is unavoidable or desirable, for instance where some EU rights, obligations or rules are repeated or described on national web pages to improve user-friendliness.

To minimise human intervention in the updating of the links to be used by the common user interface, a direct connection between the relevant technical systems of the Member States and the repository of links should, where technically possible, be established. The information included in the repository of links should be made publicly available in open, commonly used and machine-readable format, for example by APIs, to enable its reuse.

The common ICT support tools could use the Core Public Services Vocabulary (CPSV) (European Commission, 2019e) to facilitate interoperability with national service catalogues and semantics. Member States should be encouraged to use the CPSV, but are free to use national solutions.

2.3.5 ISA^2 interoperability initiatives

The ISA^2 programme entered into force on 1 January 2016 (European Union, 2015b). The ISA^2 programme supports long-standing efforts to create a European Union free from electronic barriers at national borders. ISA^2 facilitates cross-border and cross-sector interaction between European public administrations, businesses and citizens, enabling the delivery of electronic public services and ensuring the availability of common solutions, enabling them to benefit from interoperable cross-border and cross-sector public services.

The ISA^2 programme has launched many actions that are relevant to this report and to the APIs4DGov study in general. The team responsible for the ISA^2 'catalogue of public services' action (European Commission, 2018f), for example, responsible for maintaining the Core Public Service Vocabulary — Application Profile (CPSV-AP), has recently conducted a study that encourages public administrations to provide APIs that make use of the CPSV-AP data model to define data structures on which their APIs would be based (European Commission, 2019f). This would not only help them to generate their documentation automatically, but would also help them provide a catalogue of APIs or act as an API gateway. By doing so, discoverability would be enhanced and interoperability barriers would be reduced. Likewise, by making a public JSON-LD context, CPSV-AP could be reused by REST APIs publishing public services in linked data format.

Another relevant initiative addresses the 'innovative public services' (IPSs) action (European Commission, 2018g). This action aims to provide support for identifying the innovation potential and conditions of emerging disruptive technologies such as blockchain and distributed ledgers, artificial intelligence (AI) and IoT-related infrastructures, or technological solutions and platforms already mature in the private sector such as

APIs, so to better assess their impact in terms of more efficient and improved public services, as well as improved interactions between governments, citizens and business.

2.3.6 Building blocks of the 'Connecting Europe Facility'

As noted above, and to support the 'Digital Single Market' (European Commission, 2018h), the Connecting Europe Facility (CEF) funds a set of generic and reusable digital service infrastructures (DSIs), also known as 'building blocks'. The CEF building blocks offer basic capabilities that can be reused in any European or national project to facilitate the delivery of digital public services across borders and sectors. Moreover, recently, a set of pilot studies were developed to explore how CEF building blocks can support the 'Once Only Principle' (OOP) (European Commission, 2017b).

Currently, there are eight building blocks: as well as eID, the CEF offers DSIs for 'Big Data Test Infrastructure' (BDTI), 'Context Broker', 'eArchiving', 'eDelivery', 'eInvoicing', 'eSignature' and 'eTranslation'. Below, we report on the building blocks that expose APIs. Currently, the Once Only Principle (OOP) is undergoing a preparatory action within CEF. The various work packages will define if this should be considered as a Building Block or as a service of an existing building block.

- **BDTI:** this recently adopted CEF building block allows European organisations to experiment with big data technologies and move towards data-driven policymaking. It offers a range of services, technical documentation and support services that governments can use to start experimenting with their data such as a big data and analytics software catalogue, a data catalogue and data exchange APIs, onboarding and support of interested stakeholders, a big data community and a service desk (European Commission, 2019g).
- **'Context Broker':** the CEF Context Broker, developed within the FIWARE initiative, helps organisations to manage and share data in real time, describing 'what is currently happening' within their organisations, for the real-world activities they manage and for where they run their daily business processes. The CEF Context Broker aims to enable the publication of context information by entities, referred to as context producers, so that published context information becomes available to other entities, referred to as context consumers that are interested in processing the published context information. The CEF Context Broker specifications were initially based on the NGSI-9 and NGSI-10 specifications defined by the Open Mobile Alliance (OMA). That was the origin of the name of the FIWARE Context Broker API (FIWARE NGSI — next generation service interface), also referred to as CEF 'Context Broker'. The CEF Context Broker provides the FIWARE NGSI API, which is a RESTful API enabling applications to provide updates and get access to context information. The current version of the specifications of the FIWARE NGSI API are the FIWARE NGSIv2 API specifications (FIWARE Foundation, 2018). The plan is that these will evolve in line with the future ETSI NGSI for linked data (NGSI-LD) specifications, which will better support linked data (entity relationships), property graphs and semantics (exploiting the capabilities offered by JSON-LD).
- **'eDelivery':** 'eDelivery' helps public administrations to exchange electronic data and documents with other public administrations, businesses and citizens in an interoperable, secure, reliable and trusted way. The CEF eDelivery building block is based on the AS4 messaging protocol developed by OASIS (OASIS, 2013). AS4 is an open technical specification for the secure and payload-agnostic exchange of data using web services. To ease its adoption in Europe, the eDelivery building block uses the AS4 implementation guidelines defined by the Member States in the e-SENS large-scale pilot(e-SENS, 2017). This building block defines the profiles of the following standards:
  - the eDelivery AS4 profile: a modular profile of the ebXML messaging services (OASIS, 2007) and its AS4 profile specifications;
  - the eDelivery 'Service Metadata Publisher' (SMP) profile: this provides a set of implementation guidelines for the OASIS SMP specification (OASIS, 2017a);
  - the eDelivery BDXL profile: a profile of the OASIS 'Business Document Metadata Service Location specification (OASIS, 2017b);
  - the eDelivery ebCore party ID profile: a profile of the OASIS ebCore party ID type specification (OASIS, 2010).
- **eTranslation:** the main goal of the eTranslation building block is to help European and national public administrations exchange information across language barriers in the EU. It provides machine

translation capabilities to enable all DSIs to be multilingual. This building block provides a web service based on SOAP for machine-to-machine interaction. The specifications of the web service are available upon request.

2.3.7 Once Only Principle

The OOP is a core principle of the eGovernment action plan 2016-2020 (European Commission, 2016a). According to this principle, citizens and businesses should be able to provide information once and have that data shared and reused with other public administrations. Support for the OOP is generally broad, although across Europe there is a wide variation in maturity (European Commission, 2017b).

Horizon 2020 projects have directly addressed the OOP, focusing on data from business. The Stakeholder Community for Once-Only Principle (SCOOP4C) aims to investigate, discuss and disseminate how co-creation and co-production in public service provisioning for citizens can be achieved by implementing the OOP. At the end of 2018, the SCOOP4C handed over its community activities to 'The Once-Only Principle' (TOOP) project. The TOOP project aims to explore and demonstrate the OOP across borders while focusing on data from businesses. The TOOP project led to the creation of a solution architecture that connects 40 information systems using CEF building blocks, including eDelivery, eSignature and eID. A series of pilot projects have been set up all over the EU, involving 50 organisations. Pilots were selected for funding based on their cross-border relevance, potential to reduce administrative burden and feasibility of implementation (European Commission, 2019h). To highlight that, the TOOP project service design requirements are based, where possible, on the reuse of building blocks and related APIs that have proven effective in cross-border interoperability environments.

2.3.8 Common Assessment Method for Standards and Specifications

CAMSS is the European guide for assessing and selecting standards and specifications for e-government projects. Although CAMSS is not specifically focused on the API domain, some of the technical specifications and standards described in the study are covered by existing CAMSS assessments (see the references in section 2.2).

It promotes collaboration between EU Member States in defining a 'common assessment method for standards and specifications' and sharing with other countries the assessment study results for the development of e-government services.

The CAMSS assessment process and set of quality requirements are developed to align with related initiatives at European level, e.g. the EU Regulation on European standardisation (European Union, 2012).

In 2018, CAMSS was institutionalised as part of the multi-stakeholder platform's streamlined process. This is the process used to evaluate standards and technical specifications that have been proposed to be fit for use in public procurement by the European Commission, such as, for example, in the case of OASs (CAMSS Team, 2019b).

3 API glossary

This section presents the definitions of the terms collected so far as part of the APIs4DGov study. The final version of the glossary will be included in the final deliverable of the study. Different resources have been considered in compiling the glossary, including the definitions taken from standardisation bodies, the CEF (European Commission, 2019i) and the glossary previously published in (Williams, 2018).

Application programming interface (API)	An API is 'The calls, subroutines, or software interrupts that comprise a documented interface so that an application program can use the services and functions of another application, operating system, network operating system, driver, or other lower-level software program' (Shnier, 1996).
API gateway	HTTP enables the use of intermediaries to satisfy requests through a chain of connections. There are three common forms of HTTP intermediary: proxy, gateway and tunnel (Fielding and Reschke, 2014). An API gateway is a software component initially popular within the microservices world, but now also a key part of an HTTP-oriented serverless architecture. An API gateway's basic job is to be a web server that receives HTTP requests, routes the requests to a handler based on the route/path of the HTTP request, takes the response back from the handler and finally returns the response to the original client. An API gateway will typically do more than just this routing, also providing functionality for authentication and authorisation, request/response mapping, user throttling and more. Depending on the gateway features, API gateways are configured, rather than coded, which is useful for speeding up development, but care should be taken not to over use some features that might be more easily tested and maintained in code (Chaplin and Roberts, 2017).
API versioning	API versioning is one of the steps of an API life cycle (Jacobson et al., 2011). There's no common agreement on the definition of API versioning. If, from one side, an API is the embodiment of a technical contract between a publisher and a developer and this contract should stay intact, on the other side, sometimes, there is the need to start with a completely new version. So, even if we have found that API versioning is 'The ability to change without rendering older versions of the same API inoperable' (Deloitte, 2018) or that 'Non-backward-compatible changes break the API (i.e. a new one has to be released, and consumers must migrate from the old to the new one)' (Mehdi et al., 2018), we could accept the fact that, in the life of an API, starting over with a new version that might not be fully backward compatible with an older version or that might make the older version deprecated is unavoidable. Thus, retiring an API is often an unacknowledged part of the API life cycle (Boyd, 2016) and versioning is part of the API design life cycle.
Authentication	Authentication is the ability to prove that a user or application is genuinely who that person or what that application claims to be (IBM, 2014a; ENISA, 2019; NIST, 2019.
Authorisation	Authorisation protects critical resources in a system by limiting access to only authorised users and their applications (IBM, 2014c).
Backend as a service (BaaS)	BaaS is all about replacing server-side components that we code and/or manage ourselves with off-the-shelf services. … BaaS services are domain-generic remote components (i.e., not inprocess libraries) that we can incorporate into our products, with an API being a typical integration paradigm (Chaplin and Roberts, 2017).
Collaboration (on public services)	Collaboration on public services indicates that government pursues collaboration with third parties to deliver added value in public service design and/or public service delivery. Collaboration uses shared resources, taps into the power of mass collaboration on societal issues and can lead to the development of innovative, distributed and collective intelligent solutions. Collaboration is also related to the concept of service-oriented principles of reuse, composition and the modularity of

	a service. With the addition of new services, new (public) value is proposed to users. This value does not only relate to creating private value for new businesses, but also relates to creating public value, i.e. added value for society (European Commission, 2019j).
Container	An alternative to using a platform as a service (PaaS) on top of a virtual machine is to use containers (e.g. the popular Docker). Containers provide a way of more clearly separating an application's system requirements from the nitty gritty of the operating system itself (Chaplin and Roberts, 2017).
Container as a service (CaaS)	There are cloud-based services for hosting and managing/orchestrating containers on a team's behalf, often referred to as CaaS (Chaplin and Roberts, 2017).
Digital government	Digital government refers to the use of digital technologies, as an integrated part of governments' modernisation strategies, to create public value. It relies on a digital government ecosystem, comprising government actors, non-governmental organisations, businesses, citizens' associations and individuals, that supports the production of and access to data, services and content through interactions with government (OECD, 2014).
Digital platform	A digital platform is a technology-enabled business model that creates value by facilitating exchanges between two or more interdependent groups. Most commonly, platforms bring together end-users and producers to transact with each other (own elaboration).
Digital technologies	Digital technologies or ICT, include the internet, mobile technologies and devices, as well as data analytics used to improve the generation, collection, exchange, aggregation, combination, analysis, access, searchability and presentation of digital content, including for the development of services and apps (OECD, 2014).
Documentation/ definition (in API)	Documentation (or definition) is a technical content deliverable, containing instructions about how to effectively use and integrate with an API (Swagger.io, 2019a).
e-Government	This refers to the use by governments of ICT, particularly the internet, as a tool to achieve better government (OECD, 2014).
External API	An external API is an API that has been designed to be accessible outside an organisation, including by the wider population of web and mobile developers. This means that it may be used both by the developers inside the organisation that published the API and by any developers outside that organisation who may need to register for access to the interface (own elaboration).
Function as a service (FaaS)	FaaS is another form of compute as a service. FaaS is a new way of building and deploying server-side software, oriented around deploying individual functions or operations (Chaplin and Roberts, 2017).
Internal API	These APIs are generally used to facilitate the sharing of data and services between systems within an agency, avoiding the need for complex point-to-point integration. They are not visible to any person or body outside the agency and are generally in the domain of the IT department (Williams, 2018).
Internet as a service (IaaS)	IaaS allows companies to rent compute capacity — that is, a host to run their internet-facing server applications — rather than buy their own machines. It also allows them to provision hosts just in time, with the time from requesting a machine to its availability being in the order of minutes (Chaplin and Roberts, 2017).

Interoperability	This is the capability to communicate, execute programs or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units (IEEE, 1991).
Legal interoperability	Each public administration contributing to the provision of a European public service works within its own national legal framework. Legal interoperability is about ensuring that organisations operating under different legal frameworks, policies and strategies are able to work together (European Commission, 2017a).
Mashup	A mashup is a web application that uses content from more than one source to create a single new service displayed in a single graphical interface (Fichter, 2010).
Microservice	A basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of well-defined APIs, independent of any vendor, product or technology (Karmel et al., 2016).
Private API	See 'Internal API'.
Public API	See 'External API'.
Open asset	This refers to government data, software, specifications and frameworks that are open so that anyone can freely access, use, modify and redistribute their content with no or limited restrictions such as commercial use or financial charges .
Open government16/12/2019 19:13:00	Open government can be defined as the opening up of government processes, proceedings, documents and data for public scrutiny and involvement, and is now considered a fundamental element of a democratic society (OECD, 2017). The open government initiative was started in 2009 by Barak Obama (The White House, 2009); after that, numerous governments adopted open data initiatives. It is founded on the belief that greater transparency and public participation can not only lead to better policies and services, but also promote public sector integrity, which is essential for regaining the trust of citizens in the neutrality and reliability of public administrations.
Open services	These are digital public services that can be reused by other public administrations or eventually by third parties to provide value-added services via a mechanism of service composition. Open services necessitate a proper design of digital public services. The design principles of service-oriented architecture can prove useful: modular, decomposed services; interoperability through an API; and loose coupling (European Commission, 2016b).
Organisation	In general, 'Organisations' here means public administration units or any entity acting on their behalf, or EU institutions or bodies (European Commission, 2016b).
Organisational interoperability	This refers to the way in which public administrations align their business processes, responsibilities and expectations to achieve commonly agreed and mutually beneficial goals. In practice, organisational interoperability means documenting and integrating or aligning business processes and relevant information exchanged. Organisational interoperability also aims to meet the requirements of the user community by making services available, easily identifiable, accessible and user-focused (European Commission, 2017a).
Participation (in policymaking)	Participation in policymaking happens when governments open up governmental decision-making towards citizens, businesses, and public administrations to ensure an open process for participation with the aim at enhancing public value (European Commission, 2019j).

Platform as a service (PaaS)	PaaS layers on top of IaaS, adding the operating system to the infrastructure being outsourced. With PaaS only applications are deployed, and the platform is responsible for operating system installation, patch upgrades, system-level monitoring, service discovery, etc. (Chaplin and Roberts, 2017).
Public value	Public value refers to various benefits for society, which may vary according to the perspective or the actors, including the following: (i) goods or services that satisfy the needs and expectations of citizens and clients; (ii) production choices that meet citizen expectations of justice, fairness, efficiency and effectiveness; (iii) properly ordered and productive public institutions that reflect citizens' desires and preferences; (iv) fairness and efficiency of distribution; (v) legitimate use of resource to accomplish public purposes; and (vi) innovation and adaptability to changing preferences and demands (OECD, 2014).
Remote procedure call (RPC) API	A set of procedures (methods) that the client application can invoke and is executed by the server to fulfil a task. RPC APIs stem from the replacement of in-memory object messaging with cross-network object messaging in object-oriented applications (Feng et al., 2009).
Representational state transfer (REST)	A software architectural style that defines a set of constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to REST (Fielding, 2000).
RESTful API	RESTful APIs are based on the REST architectural style (Fielding, 2000).
Resource (in the REST architectural style)	In the REST architectural style, resource representation is central. 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, etc. (Fielding, 2000). 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 (Fielding, 2000).
Semantic interoperability	Semantic interoperability ensures that the precise format and meaning of exchanged data and information are preserved and understood throughout exchanges between parties, in other words 'what is sent is what is understood'. In the EIF, semantic interoperability covers both semantic and syntactic aspects (European Commission, 2017a).. — The semantic aspect refers to the meaning of data elements and the relationship between them. It includes developing vocabularies and schemata to describe data exchanges, and ensures that data elements are understood in the same way by all communicating parties. — The syntactic aspect refers to describing the exact format of the information to be exchanged in terms of grammar and format.
Serverless	Serverless covers a range of techniques and technologies. They belong to two areas: BaaS and FaaS. The key is that neither is required to manage the own-server hosts or server processes. With a fully serverless app, there is no need to think about any part of the architecture as a resource running on a host. All of the logic — whether it is own coded or whether it is integrated with a third-party service — runs within a completely elastic operating environment. State is also stored in a similarly elastic form. 'Serverless doesn't mean the servers have gone away, it means that there is no need to worry about them any more' (Chaplin and Roberts, 2017).
Service-oriented architecture	An application pattern where applications offer services to other applications by means of interfaces (European Commission, 2019j).
Smart city	There is no definitive explanation of a smart city because of the breadth of the

	technologies that can be incorporated into a city in order for it to be considered a smart city. From the definition given by Husam Al Waer and Mark Deakin in their research publication (Deakin and Waer, 2011), the factors that contribute to a city being classified as smart are: — the application of a wide variety of digital and electronic technologies in the city and its communities; — the application of ICT to enhance life and working environments in the region; — the embedding of such ICT within government systems; — the territorialisation of practices that bring the people and ICT together to foster innovation and enhance the knowledge that they offer. For a more formal definition of the term see also (Ramaprasad et al., 2017).
Software development kit (SDK)	Typically, this is a set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform or computer system (Shamsee et al., 2015).
Software ecosystem	A set of businesses functioning as a unit and interacting with a shared market for software and services, together with relationships among them. These relationships are frequently underpinned by a common technological platform and operate through the exchange of information, resources and artefacts (Messerschmitt and Szyperski, 2003).
Spatial object	A spatial object is a geographical feature, an abstract representation of a real-world phenomenon related to a specific location or geographical area (European Commission, 2019k).
Standard	A standard is a document that specifies a technological area with a well-defined scope, usually by a formal standardisation body and process (OGC, 2019e).
Technical specification	A document written by a consortium, vendor, or user that specifies a technological area with a well-defined scope, primarily for use by developers as a guide to implementation. A specification is not necessarily a formal standard (OGC, 2019e).
Transparency	Refers to disclosing relevant documents and other information on government decision-making and government activity to the general public in a way that is relevant, accessible, timely, and accurate (European Commission, 2019j).
Value chain	The value chain itself describes the full range of activities that are required to bring a product or service from conception, through the different phases of production (involving a combination of physical transformation and the input of various producer services), delivery to final consumers and final disposal after use (own elaboration).
Web application	The term 'web application' refers to a web page or collection of web pages delivered over HTTP that use server-side or client-side processing (e.g. JavaScript) to provide an 'application-like' experience within a web browser. Web applications are distinct from simple web content in that they include locally executable elements of interactivity and persistent state (W3C, 2010).
Web API	Web APIs are APIs that are offered and consumed through the web. They deliver requests to the service provider, and then deliver the response back to the requestor, i.e. they are an interface for web applications, or applications that need to connect to each other via the internet to communicate (Definition.net, 2019).
Web service	Different definitions of web service exist. The W3C defines a web service as 'a

	software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards' (W3C, 2004). This definition links the concept of a web service to a set of specific technologies (SOAP, WSDL). Others provide more generic definitions, e.g. (IBM, 2014a) states that a 'Web Service is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location'. In (Papazoglou and Georgakopoulos, 2003) the authors define a web service as 'a specific kind of service that is identified by a URI, whose service description and transport utilise open Internet standards'. These extend the W3C definitions by essentially defining a web service as a service that is offered over the web. The OASIS reference model for service oriented architecture defines a service as 'a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description' (OASIS, 2006).
Web service interface (provided by a web service)	Web service interfaces are designed to offer access to high-level functionalities for end-users (either humans or machines) (own elaboration).

4 Conclusions

APIs are a well-recognised technological solution that have been in place for many years, helping to enable interoperability and the programmatic reuse of software modules. APIs can be used in different environments (including operating systems, information systems and networks) but, in the last two decades, they have increasingly been used by applications for the exchange of their digital assets over the web. If we look back over these years, many private companies have built their success on such web APIs (e.g. Amazon, Google, Facebook, Salesforce and Twilio), giving rise to questions about their potential widespread use in the public sector.

The public sector has been changed by increasing engagement with ICT, with associated changes towards more digitally enabled services and internal workflows. Within this context, to investigate the adoption of APIs by the public sector, especially in the European Union, the European Commission launched the APIs4DGov study, with the aim of understanding (i) why governments should invest in APIs and (ii) what practical steps should be taken as a consequence. This work relates in part to this second point, as it reports on the existence of standards and technical specifications that can be used by organisations, in particular in the public sector, when adopting APIs. Standardisation is needed, as adopting ad hoc solutions may often be costly and inefficient and, above all, may block digital interoperability with the external entities of organisations — a concern for digital transformation approaches looking to increase collaboration and co-design between stakeholders.

This document gives an overview of relevant API ICT standards, technical specifications and terms. We first illustrated some definitions on the basic API concepts and maturity models. We then explained our methodology, the rationale behind the classification and some characteristics of the standards and specifications in the list of documents we collected. After that, we presented a shortlist of selected documents and European Union initiatives related to API standards and technical specifications. In the last section, we provided a glossary of the main API terms. In (Vaccari and Santoro, 2019), the reader can find the complete list of the documents we have collected and report on.

In compiling this list, we focused on API-related standards and technical specifications. We excluded some more general or very well-recognised ICT standards from the list (e.g. we did not define common terms such as HTTP, internet, etc.). In addition, while collecting these documents, we observed that, because of the rapid evolution of ICT and web technologies (particularly in the API landscape), some changes to the documents have occurred. For this reason, we also considered stability when compiling the shortlist of documents, together with use and maintenance.

One of the main challenges of this assessment was establishing the real uses of the different technical specifications and standards. This was estimated by searching through different scientific literature repositories and online documents/fora. This approach could be improved by exploring such elements *in situ* (or even *in silico*) via interviews of web API users and providers, as well as those developing and making use of online API repositories, such as the ones provided by ProgrammableWeb or RapidAPI.

We discovered that there are many aspects of APIs that have been standardised or for which specifications have been reported and used. In our research, we have found both a richness of proposals (published during an extended period) and a relatively heterogeneous set of solutions. Thus, even if a developer can choose from a number of options, sometimes it can be difficult to identify a unique solution that covers each of the aspects we have defined within this report.

It should be noted that the purpose of this work is not to suggest the adoption of particular technical specifications and standards, as it only captures a 'snapshot' of the current landscape of web APIs from, mainly, a technological point of view. Instead, the classification proposed could provide a generic context for further studies and analysis of API technical specifications and standards. Moreover, agreeing on common standards, specifications and data models should not mean changing the underlying systems already in place, but instead the adoption of APIs should facilitate interchange.

With this document, our main goal was to clarify and disseminate our understanding of the definition of key API subjects, namely the concepts, terms and list of technical specifications and standards that we have uncovered so far as part of the APIs4DGov study. We prepared it as an intermediate output of the study, to be used not only by API practitioners and developers but also by the readers who would like to know more about some of the basic API concepts, defined in the introduction. We hope that this work will further support the technical implementation of APIs by governments and that it will also contribute to updates of European

Commission reference documents on interoperability, as it provides information on many elements that could, for example, be used as part of online guidance, training or other reference information for APIs.

References

Amundsen, M., 'Web API design maturity model', 2017 (http://amundsen.com/talks/2017-07-chattanooga/index.html) (accessed 26 March 2019).

Apdex Alliance, 'Apdex overview', 2007 (http://apdex.org/overview.html) (accessed 1 July 2019).

APImetrics, 'What is the CASC score?', 2019 (https://apimetrics.readme.io/docs/what-is-the-casc-score) (accessed 1 July 2019).

BBVA, 'Products', 2019 (https://www.bbvaapimarket.com/products) (accessed 16 April 2019).

Berlin Group, 'PSD2 access to bank accounts', 2019 (https://www.berlin-group.org/psd2-access-to-bank-accounts) (accessed 13 April 2019).

Borenstein, N.S. and Freed, N., 'Multipurpose internet mail extensions (MIME) part two: media types', 1996 (https://tools.ietf.org/html/rfc2046) (accessed 4 April 2019).

Boyd, M., '2015: API deprecation and versioning now a more strategic issue', 2016 (https://www.programmableweb.com/news/2015-api-deprecation-and-versioning-now-more-strategic-issue/analysis/2016/01/04) (accessed 27 September 2019).

Bradley, J., Sakimura, N. and Jones, M., 'JSON web token (JWT)', 2015a (https://tools.ietf.org/html/rfc7519) (accessed 3 April 2019).

Bradley, J., Sakimura, N. and Jones, M., 'JSON web signature (JWS)', 2015b (https://tools.ietf.org/html/rfc7515) (accessed 3 April 2019).

CAMSS Team, 'CAMSS assessment of OAuth 2.0', 2018 (https://joinup.ec.europa.eu/release/camss-assessment-oauth-20/v100) (accessed 3 April 2019).

CAMSS Team, 'CAMSS assessment of SAML 2.0', 2019a (https://joinup.ec.europa.eu/solution/camss-assessment-saml-20/about) (accessed 3 April 2019).

CAMSS Team, 'CAMSS assessment of OAS 3.0', 2019b (https://joinup.ec.europa.eu/solution/camss-assessment-oas-30) (accessed 3 April 2019).

CEN, 'Technical specifications', 2019 (https://www.cen.eu/work/products/TS/Pages/default.aspx) (accessed 8 April 2019).

Cha, A., 'Hello, OpenAPI-to-GraphQL 1.0.0', 2019 (https://www.ibm.com/blogs/research/2019/07/openapi-graphql/) (accessed 24 July 2019).

Chaplin, J. and Roberts, M., *What Is Serverless?*, O'Reilly Media, Inc., Sebastopol, CA, 2017 (https://www.oreilly.com/library/view/what-is-serverless/9781491984178/) (accessed 7 July 2019).

Cotton, I. and Greatorex, F., 'Data structures and techniques for remote computer graphics', paper presented at International Workshop on Managing Requirements Knowledge, 9-11 December 1968, San Francisco, CA, 1968 (https://www.computer.org/csdl/proceedings/afips/1968/5072/00/50720533.pdf) (accessed 18 January 2019).

Creative Commons, 'About licences', 2019a (https://creativecommons.org/licenses/) (accessed 1 July 2019).

Creative Commons, 'Choose a License', 2019b (https://creativecommons.org/choose/) (accessed 23 July 2019).

Deakin, M. and Waer, H.A., 'From intelligent to smart cities', *Intelligent Buildings International*, Vol. 3, No 3, 2011, pp. 140–152.

Definition.net, 'Definition.net', 2019 (https://www.definition.net/privacy-policy) (accessed 14 April 2019).

Deloitte, 'API imperative: From IT concern to business mandate', *Tech trends 2018*, Deloitte Insights, 2018, pp. 111–131 (https://www2.deloitte.com/insights/us/en/focus/tech-trends/2018/api-program-strategy.html) (accessed 19 March 2018).

Department of Internal Affairs - Government of New Zealand, 'API standard and guidelines part B - Technical', 2016 (https://www.ict.govt.nz/assets/Standards-and-Compliance/API-Standard-and-Guidelines/API-Standard-and-Guidelines-Part-B-Technical-v1.0-October-2016.pdf) (accessed 1 February 2019).

EBA, *Final report —Draft regulatory technical standards on strong customer authentication and common and secure communication under Article 98 of Directive 2015/2366 (PSD2)*, EBA-RTS-2017-02, 2017 (https://eba.europa.eu/documents/10180/1761863/Final+draft+RTS+on+SCA+and+CSC+under+PSD2+(EBA-RTS-2017-02).pdf) (accessed 11 January 2019).

ECMA International, *The JSON Data Interchange Syntax*, Standard ECMA-404, 2017 (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf) (accessed 25 May 2019).

ENISA, 'Authentication Methods', 2019 (https://www.enisa.europa.eu/topics/csirts-in-europe/glossary/authentication-methods) (accessed 11 April 2019).

e-SENS, 'e-Sens — paving the way to the "live" phase of cross-border digital public services', 2017 (https://www.esens.eu/) (accessed 25 June 2019).

European Commission, 'IDABC - IDA Work Programmes', 2000 (http://ec.europa.eu/idabc/en/document/2548/3.html) (accessed 12 July 2019).

European Commission, 'IDABC - The Programme', 2004 (http://ec.europa.eu/idabc/en/chapter/3.html) (accessed 12 July 2019).

European Commission, 'INSPIRE knowledge base: Network services', 2007a (https://inspire.ec.europa.eu/network-services/41) (accessed 12 August 2019).

European Commission, 'INSPIRE knowledge base: Spatial data services', 2007b (https://inspire.ec.europa.eu/spatial-data-services/580) (accessed 12 August 2019).

European Commission, 'European eGovernment action plan 2016-2020', 2016a (https://ec.europa.eu/digital-single-market/en/european-egovernment-action-plan-2016-2020) (accessed 12 March 2019).

European Commission, *Towards faster implementation and uptake of open government : final report.*, 2016b (https://publications.europa.eu/en/publication-detail/-/publication/1071fdcc-aa45-11e6-aab7-01aa75ed71a1/language-en/format-PDF) (accessed 7 September 2017).

European Commission, 'The new European interoperability framework', 2017a (https://ec.europa.eu/isa2/eif_en) (accessed 12 March 2019).

European Commission, *EU-wide digital Once-Only Principle for citizens and businesses: Policy options and their impacts*, 2017b (https://ec.europa.eu/esf/transnationality/filedepot_download/1671/1692).

European Commission, 'New study on Digital Government APIs, APIs4DGov project', 2018a (https://ec.europa.eu/digital-single-market/en/news/new-study-digital-government-apis-apis4dgov-project) (accessed 10 July 2019).

European Commission, 'APIs4DGov API strategy survey', 2018b (https://ec.europa.eu/eusurvey/runner/APIs4DGov-Strategy) (accessed 8 April 2019).

European Commission, 'INSPIRE hackathon 2018', 2018c (https://www.plan4all.eu/inspire-hackathon-2018/) (accessed 8 April 2019).

European Commission, 'APIs4DGov study workshop: Assessing government API strategies across the EU', 2018d (https://ec.europa.eu/jrc/en/event/workshop/assessing-government-api-strategies-across-eu) (accessed 13 March 2019).

European Commission, 'The single digital gateway', 2018e (https://ec.europa.eu/growth/single-market/single-digital-gateway_en) (accessed 7 August 2019).

European Commission, 'Catalogue of services', 2018f (https://ec.europa.eu/isa2/isa2conf18/catalogue-services_en) (accessed 12 August 2019).

European Commission, *ISA2 Work Programme - 2019: Detailed description of actions part 1/2*, 2018g (https://ec.europa.eu/isa2/sites/isa/files/docs/pages/isa2_wp_2019_detailed_descriptions_part_1.pdf) (accessed 12 August 2019).

European Commission, 'Digital single market', 2018h (https://ec.europa.eu/commission/priorities/digital-single-market_en) (accessed 18 March 2019).

European Commission, *eGovernment factsheets anniversary report*, 2019a (https://ec.europa.eu/isa2/sites/isa/files/docs/news/10egov_anniv_report.pdf) (accessed 27 March 2019).

European Commission, 'JoinUp Licensing Assistant (JLA)', 2019b (https://joinup.ec.europa.eu/collection/eupl/joinup-licensing-assistant-jla) (accessed 22 July 2019).

European Commission, 'About INSPIRE', 2019c (https://inspire.ec.europa.eu/about-inspire/563) (accessed 12 August 2019).

European Commission, 'Joint Research Centre call for tenders: Facilitating access to INSPIRE data through standard-based Application Programming Interfaces (APIs)', 2019d (https://web.jrc.ec.europa.eu/callsfortender//index.cfm?action=app.tender&id=7671&home=1) (accessed 12 August 2019).

European Commission, 'Core Public Service Vocabulary', 2019e (https://joinup.ec.europa.eu/solution/core-public-service-vocabulary) (accessed 12 August 2019).

European Commission, *APIs for CPSV-AP based Catalogue of Services*, 2019f (https://joinup.ec.europa.eu/sites/default/files/news/2019-09/ISA2_APIs%20for%20CPSV-AP%20based%20Catalogue%20of%20Services.pdf) (accessed 16 September 2019).

European Commission, 'What is BDTI?', 2019g (https://ec.europa.eu/cefdigital/wiki/cefdigital/wiki/pages/viewpage.action?pageId=82773945) (accessed 28 June 2019).

European Commission, 'Improving cross-border cooperation with The Once-Only Principle', 2019h (https://ec.europa.eu/cefdigital/wiki/cefdigital/wiki/display/CEFDIGITAL/2018/09/24/Improving+cross-border+cooperation+with+The+Once-Only+Principle) (accessed 13 April 2019).

European Commission, 'Glossary - CEF Digital', 2019i (https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/CEF+Glossary) (accessed 25 March 2019).

European Commission, 'EIRA v3.0.0 overview', 2019j (https://joinup.ec.europa.eu/solution/eira/distribution/eira-v300-overview) (accessed 15 April 2019).

European Commission, 'INSPIRE glossary', 2019k (http://inspire.ec.europa.eu/glossary) (accessed 18 March 2019).

European Payment Council, *Understanding the final regulatory technical standards*, 2019 (https://www.europeanpaymentscouncil.eu/sites/default/files/infographic/2018-04/rts-infographic_April%202018.pdf) (accessed 13 April 2019).

European Union, Directive 2007/64/EC of the European Parliament and of the Council of 13 November 2007 on payment services in the internal market amending Directives 97/7/EC, 2002/65/EC, 2005/60/EC and 2006/48/EC and repealing Directive 97/5/EC, OJ L 319, 2007a, p. 1–36.

European Union, Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE), OJ L 108, 2007b, p. 1–14.

European Union, Regulation (EU) No 1025/2012 of the European Parliament and of the Council of 25 October 2012 on European standardisation, amending Council Directives 89/686/EC and 93/15/EEC and Directives 94/9/EC, 94/25/EC, 95/16/EC, 97/23/EC, 98/34/EC, 2004/22/EC, 2007/23/EC, 2009/23/EC and 2009/105/EC of the European Parliament and of the Council and repealing Council Decision 87/95/EEC and Decision No 1673/2006/EC of the European Parliament and of the Council, OJ L 316, 2012, p. 1–11.

European Union, Directive (EU) 2014/55/EU of the European Parliament and of the Council of 16 April 2014 on electronic invoicing in public procurement, OJ L 133, 2014a, p. 1–11.

European Union, Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC, OJ L 257, 2014b, p. 73–114.

European Union, Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC, OJ L 337, 2015a, p. 35–127.

European Union, Decision (EU) 2015/2240 of the European Parliament and of the Council of 25 November 2015 establishing a programme on interoperability solutions and common frameworks for European public administrations, businesses and citizens (ISA2 programme) as a means for modernising the public sector, OJ L 318, 2015b, p. 1–16.

European Union, Directive (EU) 2019/1024 of the European Parliament and of the Council of 20 June 2019 on open data and the re-use of public sector information, OJ L 172, 2019, p. 56–83.

Feng, X., Shen, J. and Fan, Y., 'REST: An alternative to RPC for web services architecture', in *2009 First International Conference on Future Information Networks*, IEEE, 2009, pp. 7–10 (http://ieeexplore.ieee.org/document/5339611/) (accessed 26 March 2019).

Fette and Melnikov, 'The WebSocket protocol', 2011 (https://tools.ietf.org/html/rfc6455) (accessed 4 April 2019).

Fichter, D., 'What is a mashup?', *Library mashups: Exploring new ways to deliver library data, information today*, Information today, Medford, NJ, 2010 (https://scholar.google.com/scholar_lookup?hl=en&publication_year=2009&pages=3-17&author=D.+Fichter&title=What+is+a+mashup) (accessed 8 July 2019).

Fielding, R.T., *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, CA, doctoral thesis, 2000 (https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm) (accessed 3 April 2019).

Fielding, R.T., 'REST APIs must be hypertext-driven', 2008 (https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven) (accessed 25 July 2019).

Fielding, R.T. and Reschke, J., 'IETF RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing', 2014 (https://tools.ietf.org/html/rfc7230) (accessed 22 July 2019).

FIWARE Foundation, 'FIWARE-NGSI v2 specification', 2018 (https://fiware.github.io/specifications/ngsiv2/stable/) (accessed 11 November 2019).

FIWARE Foundation, 'What is FIWARE?', 2019 (https://www.fiware.org/developers/) (accessed 8 July 2019).

Fowler, M., 'Richardson maturity model — step toward the glory of REST', 2010 (https://martinfowler.com/articles/richardsonMaturityModel.html) (accessed 26 March 2019).

Fowler, M. and Lewis, J., 'Microservices — a definition of this new architectural term', 2014 (https://martinfowler.com/articles/microservices.html) (accessed 14 April 2019).

Free Software Foundation, 'How to choose a license for your own work', 2018 (https://www.gnu.org/licenses/license-recommendations.en.html) (accessed 23 July 2019).

GitHub, 'The legal side of Open Source', 2019a (https://opensource.guide/legal/) (accessed 23 July 2019).

GitHub, 'Choose an open source license', 2019b (https://choosealicense.com/) (accessed 23 July 2019).

GraphQL, 'GraphQL', 2018 (https://graphql.github.io/graphql-spec/June2018/) (accessed 4 April 2019).

Hildebrand, J. and Jones, M., 'JSON web encryption (JWE)', 2015 (https://tools.ietf.org/html/rfc7516) (accessed 3 April 2019).

HL7.org, 'Welcome to FHIR', 2019 (https://www.hl7.org/fhir/) (accessed 29 July 2019).

IBM, 'What is a web service?', 2014a (https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html) (accessed 3 April 2019).

IBM, 'Identification and authentication', 2014b (https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.sec.doc/q009740_.htm) (accessed 26 March 2019).

IBM, 'Authorization', 2014c (https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.sec.doc/q009750_.htm) (accessed 26 March 2019).

IEC, 'Technical specifications (TS)', 2019 (https://www.iec.ch/standardsdev/publications/ts.htm) (accessed 4 April 2019).

IEEE, 'IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries', *IEEE Std 610*, 1991, pp. 1–217.

IETF, 'Internationalized Resource Identifiers (IRIs)', 2005 (https://tools.ietf.org/html/rfc3987) (accessed 25 May 2019).

IETF, 'Httpbis status pages', 2019 (https://tools.ietf.org/wg/httpbis/) (accessed 14 April 2019).

ISO, 'Standards', 2019 (http://www.iso.org/cms/render/live/en/sites/isoorg/home/standards.html) (accessed 8 April 2019).

ISO and IEC, 'ISO/IEC 17788:2014, Information technology — Cloud computing — Overview and vocabulary', 2014a (http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/05/60544.html) (accessed 12 November 2019).

ISO and IEC, 'ISO/IEC 17789:2014, Information technology — Cloud computing — Reference architecture', 2014b (http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/05/60545.html) (accessed 12 November 2019).

ISO and IEC, 'ISO/IEC 19086-1:2016, Information technology - Cloud computing — Service level agreement (SLA) framework — Part 1: Overview and concepts', 2016

(https://standards.iso.org/ittf/PubliclyAvailableStandards/c067545_ISO_IEC_19086-1_2016.zip) (accessed 12 November 2019).

Jacobson, D., Brail, G. and Woods, D., *APIs: A Strategy Guide*, O'Reilly Media, Inc., Sebastopol, CA, 2011 (http://shop.oreilly.com/product/0636920021223.do) (accessed 13 March 2019).

Karmel, A., Chandramouli, R. and Iorga, M., *NIST definition of microservices, application containers and system virtual machines*, NIST Special Publication 800-180 (Draft) No 180, National Institute of Standards and Technology, Gaithersburg, MD, 2016 (https://csrc.nist.gov/CSRC/media/Publications/sp/800-180/draft/documents/sp800-180_draft.pdf) (accessed 14 April 2019).

Klensin, J., Hansen, T. and Freed, N., 'Media type specifications and registration procedures', 2013 (https://tools.ietf.org/html/rfc6838) (accessed 4 April 2019).

Kotsev, A., Schleidt, K., Liang, S., Van der Schaaf, H., Khalafbeigi, T., Grellet, S., Lutz, M., Jirka, S. and Beaufils, M., 'Extending INSPIRE to the Internet of Things through SensorThings API', *Geosciences*, Vol. 8, No 6, 2018, pp. 221.

Lane, K., 'Thinking through the licensing for an API stack', 2015 (https://apievangelist.com/2015/11/07/thinking-through-the-licensing-for-an-api-stack/) (accessed 1 July 2019).

Lanthaler, M. and Gütl, C., 'On using JSON-LD to create evolvable RESTful services', paper presented at Third International Workshop on RESTful Design (WS-REST 2012), Lyon, France, 2012.

Liskin, O., Singer, L. and Schneider, K., 'Teaching old services new tricks: adding HATEOAS support as an afterthought', paper presented at Second International Workshop on RESTful Design (WS-REST 2012), Hyderabad, India, ACM, 2011, pp. 3–10 (http://ws-rest.org/2011/proc/a2-liskin.pdf) (accessed 22 March 2019).

Lutz, M., Jari, R. and Kotsev, A., 'Discussion paper on INSPIRE IR's for Network Services mapping with WFS 3.0', paper presented at 56th MIG-T meeting, 3-4 April 2019, Ispra (VA), Italy, Joint Research Centre (JRC), European Commission, 2019 (https://webgate.ec.europa.eu/fpfis/wikis/download/attachments/332217789/%5BDOC-6%5D%20IRs-WFS3.0%20mapping%20discussion%20paper%20v1.2.pdf?version=1&modificationDate=1553696665980&api=v2).

Maleshkova, M., Pedrinaci, C. and Domingue, J., 'Investigating Web APIs on the World Wide Web', in *2010 Eighth IEEE European Conference on Web Services*, IEEE, 2010, pp. 107–114 (http://ieeexplore.ieee.org/document/5693251/) (accessed 3 April 2019).

Mehdi, M., Wilde, E., Mitra, R. and Amundsen, M., *Continuous API management: Making the right decisions in an evolving landscape*, O'Reilly Media, Inc., Sebastopol, CA, 2018.

Messerschmitt, D. and Szyperski, C., *Software ecosystem: Understanding an indispensable technology and industry*, The MIT Press,Cambridge, MA, 2003 (https://mitpress.mit.edu/search?keywords=Software+ecosystem%3A+Understanding+an+indispensable+technology+and+industry) (accessed 11 November 2019).

Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M., *Microservice architecture: aligning principles, practices, and culture*, O'Reilly Media, Inc., Sebastopol, CA, 2016 (https://www.apiacademy.co/resources/books/microservice-architecture-aligning-principles-practices-culture) (accessed 26 August 2019).

Newman, S., *Building microservices: Designing fine-grained systems*, O'Reilly Media, Inc., Sebastopol, CA, 2015.

NIST, 'Authentication, CSRC Glossary', 2019 (https://csrc.nist.gov/glossary/term/authentication) (accessed 11 April 2019).

OAI, 'Open API initiative', 2019 (https://www.openapis.org/about) (accessed 28 June 2019).

OAS Community, 'Allow documenting HATEOAS APIs', 2019 (https://github.com/OAI/OpenAPI-Specification/issues/577) (accessed 4 April 2019).

OASIS, 'Reference model for service oriented architecture 1.0', 2006 (http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html) (accessed 3 April 2019).

OASIS, 'OASIS ebXML messaging services version 3.0: Part 1, core features', 2007 (http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/ebms_core-3.0-spec-os.pdf) (accessed 8 April 2019).

OASIS, 'OASIS ebCore party ID type technical specification version 1.0', 2010 (http://docs.oasis-open.org/ebcore/PartyIdType/v1.0/PartyIdType-1.0.html) (accessed 8 April 2019).

OASIS, 'AS4 profile of ebMS 3.0 version 1.0', 2013 (http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/os/AS4-profile-v1.0-os.html) (accessed 12 November 2019).

OASIS, 'Service metadata publishing (SMP) version 1.0', 2017a (http://docs.oasis-open.org/bdxr/bdx-smp/v1.0/bdx-smp-v1.0.html) (accessed 8 April 2019).

OASIS, 'Business document metadata service location version 1.0', 2017b (http://docs.oasis-open.org/bdxr/BDX-Location/v1.0/BDX-Location-v1.0.html) (accessed 8 April 2019).

OData, 'OData - the best way to REST', 2019 (https://www.odata.org/) (accessed 4 April 2019).

OECD, 'Recommendation for Digital Government Strategies', 2014 (http://www.oecd.org/gov/digital-government/recommendation-on-digital-government-strategies.htm) (accessed 2 February 2019).

OECD, 'Open Government', 2017 (http://www.oecd.org/gov/open-government.htm) (accessed 19 March 2019).

OECD, *Strengthening digital government*, OECD Going Digital Policy Note, OECD, Paris, 2019 (http://www.oecd.org/going-digital/strengthening-digital-government.pdf) (accessed 2 April 2019).

OGC, 'GeoXACML implementation specification', 2011 (https://www.opengeospatial.org/standards/geoxacml) (accessed 25 May 2019).

OGC, 'OGC Web Feature Service 3.0: Part 1 - Core', 2018a (https://cdn.rawgit.com/opengeospatial/WFS_FES/3.0.0-draft.1/docs/17-069.html) (accessed 14 April 2019).

OGC, 'OGC seeks public comment on WFS 3.0 candidate standard', 2018b (https://www.opengeospatial.org/pressroom/pressreleases/2830) (accessed 14 April 2019).

OGC, 'OGC Standards and Supporting Documents', 2019a (http://www.opengeospatial.org/standards) (accessed 28 June 2018).

OGC, 'OGC history (detailed)', 2019b (http://www.opengeospatial.org/ogc/historylong) (accessed 8 August 2019).

OGC, 'An open standard draft for querying geospatial information on the web.: opengeospatial/WFS_FES', 2019c (https://github.com/opengeospatial/WFS_FES) (accessed 14 April 2019).

OGC, 'OGC invites developers to the OGC API hackathon', 2019d (http://www.opengeospatial.org/pressroom/pressreleases/2981) (accessed 14 April 2019).

OGC, 'Glossary of Terms - S', 2019e (https://www.opengeospatial.org/ogc/glossary/s) (accessed 8 April 2019).

Open Banking, 'Open banking standards', 2019 (https://www.openbanking.org.uk/providers/standards/) (accessed 13 April 2019).

Open Source Initiative, 'Frequently answered questions', 2019 (https://opensource.org/faq#which-license) (accessed 23 July 2019).

Papazoglou, M.P. and Georgakopoulos, D., 'Introduction: Service-oriented computing', *Communications of the ACM*, Vol. 46, No 10, 2003, pp. 24–28.

PolishAPI, 'About PolishAPI', 2019 (https://polishapi.org/en/#about) (accessed 13 April 2019).

Postman, 'The collaboration platform for API development', 2019 (https://www.getpostman.com) (accessed 8 November 2019).

ProgrammableWeb.com, 'Search the largest API directory on the web', 2019 (https://www.programmableweb.com/category/all/apis) (accessed 18 March 2019).

PWC, *The communication between third party providers and banks: What will the impact of technology be?*, PSD2 in a Nutshell, No 2, 2016 (https://www.pwc.com/it/en/industries/banking/assets/docs/psd2-nutshell-n02.pdf) (accessed 13 April 2019).

Ramaprasad, A., Sánchez-Ortiz, A. and Syn, T., 'A unified definition of a smart city', in *Janssen, M., Axelsson, K., Glassey, O., Klievink, B., Krimmer, R., Lindgren, I., Parycek, P., Scholl, H. J. and Trutnev, D. (eds)*, Electronic Government: 16th IFIP WG 8.5 International Conference, EGOV 2017, St Petersburg, Russia, September 4-7, 2017, Proceedings, Cham, Switzerland, Springer, 2017, pp. 13–24.

RapidAPI, 'API marketplace - Free public & open REST APIs', 2019 (https://rapidapi.com/) (accessed 28 June 2019).

Semantic Integration, 'HyperGraphQL', 2019 (https://www.hypergraphql.org/) (accessed 24 July 2019).

Shamsee, N., Klebanov, D., Fayed, H., Afrose, A. and Karakok, O., *CCNA data center DCICT 640-916 official cert guide*, Cisco Press, 2015.

Shnier, M., *Dictionary of PC Hardware and Data Communications Terms*, O'Reilly Media, Inc., Sebastopol, CA, 1996.

Singleton, A., 'The economics of microservices', *IEEE Cloud Computing*, Vol. 3, No 5, 2016, pp. 16–20.

SPDX Workgroup-Linux Foundation, 'Software package data exchange (SPDX)', 2019 (https://spdx.org/) (accessed 22 July 2019).

STET, 'Stet : PSD2', 2019 (https://www.stet.eu/index.php?id=49) (accessed 13 April 2019).

Swagger.io, 'What is API documentation, and why it matters?', 2019a (https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/) (accessed 26 March 2019).

Swagger.io, 'About swagger', 2019b (https://swagger.io/about/) (accessed 1 September 2018).

Swedish Governmental Agency for Innovation Systems, 'About the project - Swedish API license', 2019 (http://apilicens.se/en/om/projektet/) (accessed 12 September 2018).

Taelman, R., Vander Sande, M. and Verborgh, R., 'GraphQL-LD: Linked data querying with GraphQL', paper presented at the 17th International Semantic Web Conference, 8-12 October 2018, Monterey, CA, 2018 (https://comunica.github.io/Article-ISWC2018-Demo-GraphQlLD/) (accessed 12 November 2019).

The White House, 'Open Government Initiative', 2009 (https://obamawhitehouse.archives.gov/open) (accessed 19 March 2019).

UK Government, 'Performance: Find performance data of government services', 2019 (https://www.gov.uk/performance) (accessed 1 July 2019).

Vaccari, L. and Santoro, M., *API standards and technical specifications - APIs4DGov*, European Commission, Joint Research Centre (JRC), 2019 [Dataset], PID: http://data.europa.eu/89h/5a431f38-1e2c-449a-898e-34f2a3234c3b.

W3C, 'Web services architecture: W3C working group note 11 February 2004', 2004 (https://www.w3.org/TR/ws-arch/) (accessed 3 April 2019).

W3C, 'Mobile web application best practices', 2010 (https://www.w3.org/TR/mwabp/#webapp-defined) (accessed 19 April 2019).

W3C, 'SPARQL 1.1 query language', 2013a (https://www.w3.org/TR/sparql11-query/) (accessed 11 April 2019).

W3C, 'SPARQL 1.1 update', 2013b (https://www.w3.org/TR/sparql11-update/) (accessed 11 April 2019).

W3C, 'SPARQL 1.1 protocol', 2013c (https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/) (accessed 11 April 2019).

W3C, 'SPARQL 1.1 graph store HTTP protocol', 2013d (https://www.w3.org/TR/sparql11-http-rdf-update/) (accessed 11 April 2019).

W3C Community Group, 'JSON-LD primer', 2019 (https://json-ld.org/primer/latest/) (accessed 11 August 2019).

Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D. and Gurevich, Y., 'Explicating SDKs: uncovering assumptions underlying secure authentication and authorization', paper presented at 22nd USENIX conference on Security Symposium, 14-16 August 2013, Washington DC, 2013 (http://www.cs.virginia.edu/~evans/pubs/usenix2013/explicating.pdf) (accessed 12 November 2019).

Wilde, E., 'Surfing the API Web: Web Concepts', in *WWW'18 Companion*, The 2018 Web Conference companion, April 23-27, 2018, Lyon, France, ACM, New York, NY, 2018, pp. 797–802 (https://doi.org/10.1145/3184558.3188743) (accessed 30 August 2019).

Wilde, E., 'Web Concepts', 2019 (http://webconcepts.info/) (accessed 30 August 2019).

Williams, M., *Digital Government Benchmark - API study*, Joint Research Centre (JRC), European Commission, Ispra (VA), Italy, 2018 (https://joinup.ec.europa.eu/document/digital-government-benchmark-api-study) (accessed 19 March 2019).

Wittern, E., Cha, A. and Laredo, J.A., 'Generating GraphQL-Wrappers for REST(-like) APIs', in *Mikkonen, T., Klamma, R. and Hernández, J. (eds), Web Engineering: 18th International Conference, ICWE 2018, Caceres, Spain, June 5-9, 2018, Proceedings*, Springer, Cham, Switzerland, 2018, pp. 65–83 (http://arxiv.org/abs/1809.08319) (accessed 24 July 2019).

List of abbreviations and definitions

AISP	account information service provider
API	application programming interface
APIs4DGov	application programming interfaces for digital government
ASPSP	account servicing payment service provider
BaaS	backend as a service
CAMSS	common assessment method for standards and specifications
CEF	Connecting Europe Facility
CEN	European Committee for Standardization
CaaS	container as a service
CPSV	Core Public Services Vocabulary
CPSV-AP	Core Public Services Vocabulary — Application Profile
DG	Directorate-General
DSI	digital service infrastructure
eID	electronic identification
EBA	European Banking Authority
EIF	European interoperability framework
ENISA	European Union Agency for Cybersecurity
ETSI	European Telecommunications Standards Institute
EU	European Union
FaaS	function as a service
FIWARE	future internet ware
HAL	Hypertext Application Language
HATEOAS	hypermedia as the engine of application state
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	internet as a service
IANA	Internet Assigned Numbers Authority
ICT	information and communications technology

IDA	Interchange of Data between Administrations
IDABC	interoperable delivery of European eGovernment services to public administrations, business and citizens
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
INSPIRE	infrastructure for spatial information in Europe
IoT	internet of things
IRI	internationalised resource identifier
ISA/ISA^2	interoperability solutions for public administrations, businesses and citizens
ISO	International Organization for Standardization
JLA	JoinUp Licensing Assistant
JRC	Joint Research Centre
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
NGSI	next generation service interface
NIST	National Institute of Standards and Technology
OAS	OpenAPI specification
OASIS	Advancing Open Standards for the Information Society
OGC	Open Geospatial Consortium
OOP	Once Only Principle
OP	OpenID provider
PaaS	platform as a service
PAP	policy administration point
PDP	policy decision point
PEP	policy enforcement point
PIP	policy information point
PISP	payment initiation service provider
PSD	Payment Services Directive
PSD2	revised Payment Services Directive

PSI Directive	Public Sector Information Directive
PSP	payment service provider
RAML	RESTful API Modeling Language
RDF	resource description framework
REST	representational state transfer
RP	relying party
RPC	remote procedure call
RTS	regulatory technical standards
SAML	Security Assertion Markup Language
SCA	strong customer authentication
SCOOP4C	Stakeholder Community for Once-Only Principle
SDK	software development kit
SLA	service-level agreement
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SPDX	Software Package Data Exchange
TOOP project	the Once-Only Principle project
TPP	third-party payment service provider
URI	uniform resource identifier
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language
W3C	World Wide Web Consortium
WFS	Web Feature Service
WSDL	Web Service Description Language
YAML	YAML Ain't Markup Language

List of boxes

List of figures

List of tables

GETTING IN TOUCH WITH THE EU

In person

All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: https://europa.eu/european-union/contact_en

On the phone or by email

Europe Direct is a service that answers your questions about the European Union. You can contact this service:

- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),

- at the following standard number: +32 22999696, or

- by electronic mail via: https://europa.eu/european-union/contact_en

FINDING INFORMATION ABOUT THE EU

Online

Information about the European Union in all the official languages of the EU is available on the Europa website at: https://europa.eu/european-union/index_en

EU publications

You can download or order free and priced EU publications from EU Bookshop at: https://publications.europa.eu/en/publications. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see https://europa.eu/european-union/contact_en).

KJ-NA-29984-EN-N

The European Commission's science and knowledge service

Joint Research Centre

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub

ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

EU Science, Research and Innovation

EU Science Hub

Publications Office of the European Union

doi:10.2760/675

ISBN 978-92-76-13183-0