Call for Tenders / Ref. 2151-904-31

Contracting Institute
European Trade Union (ETUI)
Research department
Boulevard du Roi Albert II n°5, box 4
B-1210 Brussels
Belgium
(AISBL 0418.812.841)

For the Project

European Works Councils database – new backend and data architecture
2151-904-31

Expertise required:

This project seeks to implement a production-ready CRUD system using state-of-the-art front-end frameworks and develop a back-end pilot infrastructure that is focused on existing research data. This infrastructure should comply with the FAIR principles of research data management and follow the W3C recommendations for transparent RESTful APIs in social sciences data. In order to create a successful API, developers need to possess knowledge about API frameworks for data sharing, design and architecture, messaging and other services. They should have a thorough understanding of the REST architectural style and its use of HTTP. RESTful systems support different formats, such as HTML, YAML, XML and JSON, but JSON is recommended as it is a lightweight, easy way to exchange data and maintain its structure. Finally, authentication frameworks is a plus, it is intended to connect OAuth1 for making data and resources accessible, secure and easily integrated. The developers should be confident to explore other new approaches that might be appropriate for the organisation.

The developers in charge of this should also possess professional communication skills in order to collaborate efficiently together with the researchers and users, as well as in order to liaise with other interested parties. Consistent and precise documentation skills are essential to maintain the understandability and adaptability of the API for other developers and data consumers. Tools like Docusaurus2, Raml3, or Apiary4 are available to assist with documentation. However, Swagger5 is recommended in this case (section 3.2 of the attached report).

Furthermore, the team in charge need to have a solid grasp of security (in order to prevent data breaches through hacked, exposed or broken APIs), cloud platforms (AWS is explored in the attached report) and testing tools, such as SmartBear6, API Fortress7, Rspec8, API Science9, Parasoft10, Postman11, or Runscope12 (in order to test endpoints and check responses).

1 https://oauth.net/
2 https://docusaurus.io/
3 https://raml.org/
4 https://apiary.io/
5 https://swagger.io/
6 https://smartbear.com/
7 https://apifortress.com/
8 https://rspec.info/
9 https://www.apiscience.com/
10 https://www.parasoft.com/
11 https://www.postman.com/
12 https://www.runscope.com/
In the tender, the potential candidates should demonstrate acquired expertise in developing backend solutions for online SQL databases and API interfaces. A list of previous projects completed in this area should be attached to the submissions.

**Background information on the project**

1. The European Trade Union Institute (ETUI) has been curating a database of European Works Councils since 1999. Since then the database has been developed by an in-house expert and data manager and expanded by new modules, data and functions. Since 2010 it also feeds directly into a dedicated website www.ewcdb.eu. The contents of the database are stored in a MySQL format, while it uses MS Access as a graphical user interface (GUI) to feed and query the data. The database has several contributors connecting to it via a Microsoft ODBC connector.

2. For numerous reasons, the ETUI seeks to review the EWCdb architecture and technical solutions used to feed, analyse and work with the contents of the database. In future the ETUI also seeks possible solutions to automate the analysis of agreements according to set criteria. The ETUI also seeks possible options in having other (restricted) external parties accessing and changing parts of the data, possibly with validation by database managers/admin.

3. The ETUI collects and stores an important amount of both public and confidential information from all across the EU, information which is highly valuable to multiple different parties. As a research and training centre, it needs to ensure data is secure but accessible to optimise the insights derived from it.

4. To this end the ETUI commissioned an external report analysing the current architecture and its features as well as proposing solutions for a more modern, flexible and robust data architecture and backend to be developed in future.

**Description of tasks to be performed by the contractor:**

With this Call for Tender the ETUI seeks to implement the recommendations of the attached expert report into practice, excluding the automated natural language programming (NLP) agreements analysis at this stage (will be developed / implemented in future).

Specifically, the contractor will be expected to develop solutions based on the recommendations in the Expert Report in the following areas:

1. Development of the new database architecture
   1.1. A RESTful infrastructure combined with a CRUD interface for users;
   1.2. Development of a RESTful API (preferably according to the Rapid API standards) for services-oriented data infrastructure, including development of a full API documentation with an embedded UI on the main database reference website;

   1.2.1. The frontend framework includes the services used to create the user experience, including user functionality and the interface, and involves caching, wireframing, synchronization and UI development.
1.2.2. The backend framework includes the database server, involving data storage and integration, server-side logic, versioning and push actions. These two frameworks are connected via the API. In the context of this project, the recommended backend framework is discussed in 3.2 and 3.4 of the attached report where ExpressJS and AWS Amplify are connecting middleware, and the recommended frontend framework is ReactJS.

1.3. Connecting the API to a CRUD application (such as ReactJS or NodeJS) to serve as a GUI for the database;

1.4. Serverless CRUD application implementation, preferably Amplify by Amazon Web Services

1.5. Design and implementation of a reliable and fast search engine including application of log history of previous searches (GET requests);

1.6. Connection (native connectivity) to visualization frameworks: linkage and configuration of the connection to D3.js and Tableau as a GUI interfaces to analyse and visualise data in the database as well as making the architecture Plotly-Dash-ready.

1.7. Native connectivity to mobile and web apps.

1.8. Future connectivity readiness to automated content analysis of agreements stored in the database based on Natural Language Programming (NLP) and other technologies.

To illustrate, the sought infrastructure to be developed is:

*Fig. 1. Overview of the planned Services-Oriented Data infrastructure for the EWCdb*

2. Transfer the currently available database into the new architecture as well as test and troubleshoot until a fully-operational state.

3. In carrying out the contract, the Contractor will be expected to liaise with the company developing on behalf of the ETUI the new [www.ewcdb.eu](http://www.ewcdb.eu) website (Drupal 9) as well as with the ETUI’s internal IT department.

4. Prepare full technical documentation of the new database

5. Provide training to the ETUI staff (3-4 people max) on handling the new architecture along with reference material for further autonomous use by the ETUI.
**Estimated completion time**

The bidder is expected to propose a timeline for the completion of the project. It is estimated that frontend and backend infrastructures are each expected to take about two to six months to be developed each (although they can be developed simultaneously by different developers), and then approximately two to three weeks should be budgeted for launching and testing.

**GDPR rules**

The candidate / tenderer is required to comply with the requirements of the General Data Protection Regulation (GDPR), ensuring the processing, data security and data protection of data subjects when personal data are involved in the data protection in the framework of a future contract or intention to conclude a contract.

Further information on how the ETUI handles personal data and contact details is available on the privacy notice on the ETUI website at www.etui.org.

**Duration and value of the tender**

The ETUI plans to conclude a contract with the successful bidder and award a contract from September 2021. Bidders should propose an indicative timeline for milestones and deliverables. The final deadline for all the aforementioned deliverables is 15 May 2022, but earlier completion of the task will be favoured.

The price indicated should include VAT and all taxes and shall cover the work performed by the bidder to complete the tasks listed in the description of the tender. Cost of software needed to be purchased by the ETUI should be indicated, but should not be included in the price.

**Award criteria:**

The bid will be awarded on the basis of “best value for money”, especially based on the quality of previous work or services rendered. The criteria will be assessed based on the quality of the bid documentation and the experience of the bidder (to be included in the bid documentation).

**Deadline for the submission of tenders:**

Candidates should submit their bid in electronic form to Nicola Countouris, Director of the Research Department of the ETUI (email: skasiers@etui.org) and Romuald Jagodziński, Senior Researcher at the ETUI (email: rjagodzinski@etui.org) at the latest by **22 September 2021**.
**Terms and conditions:**

1. The expertise requires a specialist with in-depth knowledge of the project topic and understanding of the nature of the service provided by the ETUI. The bid must include a C.V. of the bidder, including his/her experience and record of relevant projects for the task described.

2. The overall value of the tender is maximum 50000 Euro (taxes and charges included). The price includes the work performed by the bidder to complete the tasks listed above and provide the specified deliverables.

   The payment will be made in three stages: 20% of the contracted amount after the signature of contract; 40% after delivery of first deliverables agreed between the successful bidder and the ETUI; 40% after the delivery of the completion of the project and no later than 15/05/2022.

3. The ETUI and the successful contractee share the right to use all intellectual property which the contractee generates in fulfilment of the contract, the material results of which will be protected by copyright.

4. The winning bidder shall accept responsibility for any legal obligations entailed by the contract and is in particular required to provide his/her country of origin the requisite tax statements relating to the services supplied.

5. Value tax, where applicable, shall be shown separately on all invoices as a net extra charge. All other taxes, levies and expenses shall be borne by the contractor and shall not be included in the relevant invoices.

6. The winning bidder accepts all reporting obligations to the EU to which the ETUI is subject and in case of audit will cooperate to provide necessary information.

7. In case of disputes the Belgian courts will be competent.

The ETUI is financially supported by the European Union

---

**Annex 1:** Declaration of Honour

**Annex 2:** Expert Report prepared for the European Trade Union Institute (ETUI) On the European Works Councils Database Architecture and its Data Services
ANNEX 1 TO NOTICE

REF.: ETUI call for tender N° 2051-904-31

Annex 1 - Declaration on Honour
(Selection criteria, point 6 of the notice)

I HEREBY CERTIFY, AS CANDIDATE/LEGAL REPRESENTATIVE OF THE CANDIDATE,

THAT THE CANDIDATE:

- is not bankrupt or being wound up, is not having his/her affairs administered by the courts, has not entered into an arrangement with creditors, has not suspended business activities, is not the subject of proceedings concerning those matters or is not in any analogous situation arising from a similar procedure provided for in national legislation or regulations;

- has not been convicted of an offence concerning his/her professional conduct by a judgment which has the force of res judicata;

- is not guilty of grave professional misconduct proven by any means which the contracting authority can justify;

- has fulfilled his/her obligations relating to the payment of social security contributions or the payment of taxes in accordance with the legal provisions of the country in which he/she is established or with those of the country of the contracting authority or those of the country where the contract is to be performed;

- has not been the subject of a judgment which has the force of res judicata for fraud, corruption, involvement in a criminal organisation or any other illegal activity detrimental to the Communities’ financial interests;

- is not currently subject to an administrative penalty referred to in Article 96(1) of the Financial Regulation;

Signature ........................................ Date:

THAT THE CANDIDATE/legal representative of the candidate:

- is not subject to any conflict of interest;

- that the candidate will inform the contracting authority, without delay, of any situation constituting a conflict of interest or which could give rise to a conflict of interest;

Signature ........................................ Date:

To the best of my knowledge, all information provided is true and accurate.

Signature ........................................ Date:

The ETUI reserves the right to check the information provided. Together with this form, duly signed, the candidate undertakes to send any additional document which the ETUI considers necessary to perform its checks.

By signing this form, the undersigned acknowledges that he/she accepts controls/audits from the European Commission under the same conditions as the ETUI.

Signature ........................................ Date:
Annex 2:

Expert Report prepared for the European Trade Union Institute (ETUI)
On the European Works Councils Database Architecture
and its Data Services

Pedro V. Hernández Serrano
April 2021

Disclaimer: No person’s data has been used for this research; thus, no GDPR or other legal obligations regarding personal privacy and security have been involved. The technologies mentioned are recommendations provided by the consultant, who has no commercial association with any of the products mentioned, no intellectual ownership over the technologies.
# Table of Contents

Executive Summary 3
List of abbreviations 3
1. Introduction 4
   1.1. Background 5
   1.2. Motivation and Challenges 6
   1.3. Value Proposition 7
2. Diagnosis 8
   2.1. Technologies 8
   2.2. Users 10
   2.3. Services 11
   2.4. Diagnosis Conclusion 12
3. Using a RESTful API as a services-oriented data infrastructure 13
   3.1. What is an API? 13
   3.2. Why is an API suitable for EWCdb? 15
   3.3. What is a CRUD application? 18
   3.4. The EWCDB Portal: a CRUD app implementation 19
   3.5. Proposed Data Infrastructure 21
   3.6. Requirements 23
   3.7. Alternative Technologies 24
4. Services and Use Cases 25
   4.1. Connection to website and search engine 25
   4.2. Connection to Visualization frameworks 26
     4.2.1. Data Visualization with D3 27
     4.2.2. Data Visualization with Plotly-Dash 28
   4.3. Connecting Tableau 30
     4.3.1. Options and bridges 30
     4.3.2. Tableau special features 32
   4.4. Data Analytics & Automated analysis of agreements 34
5. Final Remarks 35
Executive Summary

The following report presents a detailed overview of the current data architecture of the EWCdb and diagnoses several challenges, strengths and opportunities for the future of the infrastructure. This report summarises the state-of-the-art technologies more suitable for a smooth adoption given ETUI background. Additionally, this report gives detailed recommendations and provides insights for future development opportunities.

List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CI/CD</td>
<td>Continuous Integration / Continuous Deployment</td>
</tr>
<tr>
<td>ETUC</td>
<td>European Trade Union Confederation</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>EWC</td>
<td>European Works Council</td>
</tr>
<tr>
<td>EWCdb</td>
<td>European Works Council Database</td>
</tr>
<tr>
<td>FAIR</td>
<td>Findable Accessible Interoperable Reusable</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>MNC</td>
<td>Multinational Company</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>SCE</td>
<td>Societas Cooperative Europaea</td>
</tr>
<tr>
<td>SE</td>
<td>Societas Europaea</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>
In the current digital sphere, there exist vast quantities of data. The European Trade Union Institute (ETUI) itself collects and stores an important amount of both public and confidential information from all across the EU, information which is incredibly valuable to multiple different parties. As a research and training centre, it needs to ensure data is secure but accessible to optimise the insights derived from it.

Also, databases have become a pivotal point for research and development, therefore, different actors are advancing on state-of-the-art technologies, enabling the database to be efficient in its operations, secure and comprehensive. Nowadays, one can find use cases like LSE’s Climate Change Laws of the World, which more recently are referred to as knowledge databases. These types of databases are designed for consumption and knowledge generation, readable and accessible for humans and machines by design, as opposed to more traditional paradigms, where databases would be storage-centred minded. In this example, LSE’s Climate Change Laws of the World provide a stimulating, easily navigable and comprehensive collection of all laws, policies and litigation cases about the climate.

In this report, the current architectural design of the ETUI database is examined. Moreover, recommendations are provided for updating and improving its design, aiming to signal the direction towards an end-to-end knowledge database. In addition, important concepts of services-centred architecture are discussed such as REST API for data consumption protocols, as well as their importance in the light of scalable research infrastructure and maintenance.

Finally, this report explores immediate use cases on which ETUI research team can benefit from having an updated architecture implemented.

1.1. Background

The ETUI is the independent research and training centre of the European Trade Union Confederation (ETUC) which itself affiliates European trade unions into a single European umbrella organisation. The ETUI created and has been curating the database of European Works Councils since 1999 called EWCdb. This database identifies multinational companies and agreements that fall within the scope of the EWC Directive. This database has been maintained by an in-house expert and data manager, and from 2006 onwards can be accessed via a web portal at www.ewcdb.eu. Finally, it underwent a major update most recently in 2015.

The EWCdb aims to continue: 1) monitoring EWC and SE Works Council as well as the SCE (Societas Cooperative Europaea, European Cooperative Society) population; 2) analysing evolutions of EWCs, SE works councils and SCEs; 3) providing information to practitioners, policy makers and researchers on EWCs.

The content in the database is composed of companies (multinational companies, MNCs) having (one or more) European Works Councils. All these

1 https://www.etui.org/
2 https://climate-laws.org/
objects are stored in a MySQL database\(^3\) implementation hosted on premise (in-house). Aligning to the database aims - **monitor, analyse and publish** - the database therefore considers users that will advance those aims. Currently, those users interact with the database using MS Access\(^4\) as a graphical user interface (GUI) combined with Microsoft ODBC connector\(^5\).

During the last decade, we have experienced a quick and drastic evolution of databases, as the data became a commodity, an asset, which normally would be the centre of entire business models; therefore the related technologies for accessing the data have evolved greatly. Nowadays cutting-edge databases implementations, in industry, government and research rely on the so-called API Economy\(^6\) build on top of knowledge databases.

### 1.2. Motivation and Challenges

Beyond the obvious technical reasons for moving to a more robust data infrastructure, the greatest driver for evolving is the future potential usage of the database itself. The database shall be seen as a data product for research (knowledge database) instead of simply information stored in a server. These so called knowledge databases (a type of data products) are normally structured in a way that they become web-services oriented software.

The biggest challenge therefore would be to define the business model of the knowledge database in the long run, which might include all potential technologies to be integrated, potential users, together with the type of services planned to be provided. Part of this current challenge is the current capacity of the database administration officials.

On the other hand, technical motivations are quite clear, the database users have been using MS access as a proxy for UI communication to the MySQL database using a standard ODBC connector. An ODBC connection assumes that the credentials for the users are controlled in the database side by the database manager, which is rather inconvenient given that the users might need very specific rights to access specific parts of the database (e.g. validation). Moreover, it is known that Microsoft will stop supporting and developing technologies for the ODBC connector; therefore, security aspects might be an issue, since no future patches will be considered.

Furthermore, the ETUI also aims on having external parties accessing and editing the database; therefore, different levels of permission are needed. But most importantly, a main objective of moving to a more scalable architecture is to natively have the ability to connect to ad-hoc software or systems that would automate the analysis of EWC agreements and cases, additionally making

\(^3\) https://www.mysql.com/
\(^4\) https://www.microsoft.com//microsoft-365/access
\(^5\) https://docs.microsoft.com/power-query/connectors/odbc
possible compatibility frameworks with data analytics tools that have the potential to give the ETUI the means to advance pending research with cutting edge technologies, combining the advantages of proprietary software and open source technologies for web applications and data science.

1.3. **Value Proposition**

Certainly, a full re-engineering process of the current system is not exactly the answer, rather a change in paradigm when managing the data, going from a horizontal data access model to services-oriented data architecture.

The current relational database model which the data relies on, is in fact the most adequate for the type of operations and content, the MySQL server solution is good. However, relying on the database for all the weightlifting of services is rather unscalable and not exactly recommended given the database is seen as a knowledge base. Therefore a way to tackle this is following best practices.

It is proposed to develop a RESTful API on top of the database, lifting up all the burden of external services and a variety of current and future applications.

The proposed paradigm offers a number of advantages:

- Possibility to integrate simple serverless applications like a user’s management system
1. Tailored made security aspects controlled from the API never accessing the DB
2. Virtually no limit on applications and services to be connected, e.g. a website, search engine, web app, mobile app, annotator system, analytics tool, visualization portals, dashboards, etc.
3. Possibility to document it as Semantic interoperability services\(^7\) compiling with the European Commission recommendation on data usage.

Having a robust infrastructure is crucial, state of the art databases rely on serverless architectures, cloud solutions, or graph-based databases, and all use APIs as a middleware data communication manager.

1. The following section discusses the current status of the ETUI workflow for the EWCdb in terms of technologies, users and services.

2.1. **Technologies**

The currently existing technological components are a Database system, a user-data communication system and a website engine, these are described as follow:

**Database system:** Relational database implementation in MySQL server. Currently composed by 173 Tables and 1 view (see figure 3). The server implementation is on premise, meaning that it is managed and maintained by in-house database administrators. It is recognized that the majority of the tables and catalogues are merely operational and, to a greater extent, experimental. Only three main tables related to agreements; cases and unions are the pivotal focus in the data schema.

User-data communication system: MS Access Client. Currently serves as a content management input and editing interface connecting to the database and only available as an internal solution.

Website engine: MicroPHP framework. Currently serves as a querying engine to the database and provides the EWCDB website data. More discussion about the current website is in section 4.1

<table>
<thead>
<tr>
<th>System</th>
<th>Technology</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database system</td>
<td>MySQL server</td>
<td>Reliable, Robust, Domain expertise</td>
<td>Classical disadvantages when comparing with NoSQL. Do not support semantic relations</td>
</tr>
<tr>
<td>User-data communication system</td>
<td>MS Access Client</td>
<td>Low learning curve, No tailored development needed</td>
<td>Generic permissions to users</td>
</tr>
<tr>
<td>Website engine</td>
<td>MicroPHP framework</td>
<td>Lightweight implementation</td>
<td>If there are changes in the data, the connectors shall change also</td>
</tr>
</tbody>
</table>

Table 1: Overview on current technologies.
2.2. **Users**

We can define a user of the EWCdb as an individual or group that will manipulate the information in the database in two ways: for **contribution** and for **consumption**. The data manipulation relies on a user interface system that connects the user with the data.

- **User role: Contributor**

  A contributor of the database is that individual that has credentials to edit or curate the agreements or any other instance contained in the database. A contributor will perform additions to the database, and most importantly keeps the quality and accuracy of the information. They tackle two of the EWCdb aims: monitoring and analysis.

- **User role: Consumer**

  A consumer of the database is that individual that has credentials to query and explore the agreements or any other instance contained. A consumer will perform requests to the database, and these users are normally external parties. Some examples of the type of users are the following:

  - **External researchers or academics (individuals or teams):**
    - Researchers that request tailored queries, as well as research projects to which the ETUI contributes.
    - Researchers who ETUI subcontracts to perform tailored tasks: e.g., analysis of the EWC agreements, analysis of EWC court cases, analysis of samples of the EWC population.
    - Researchers with whom ETUI collaborates on scientific output and publications.
  
  - **ETUFs and national trade union officials**
    - Some ETUFs have already given up running their own EWC database and rely entirely on the ETUI (e.g., EFBWW).
    - Some other ETUFs are candidate although technical limitations place them on hold.
    - EWC coordinators from national unions.
    - Any other external union.
    - Unions that try to find information to target companies.

  - **Commercial interest**
    - ETUC advisors having access to our data on EWCs and SEs.
    - Trade union consultants.
    - Legal offices who work for companies/EWCs and search for agreements of the company and others.
    - Individual users.

  - **Parties with general interest**
    - TU newsletters searching for news (IR share, liaisons sociales).
    - Users currently querying the database via the website, limited to keyword search.
    - The general public.
2.3. **Services**

In this report, the term services is coined after a generalization made when software applications were described, i.e., all applications are software, but not all the applications are services. A software is a service consumed by a user where the object that is consumed is structured data (in our case).

There is a number of existing services that have been identified, some of which are currently in use, but we also consider envisioned services additional to the existing ones.

<table>
<thead>
<tr>
<th>Existing Services</th>
<th>Additional Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Website and UI for database search</td>
<td>● Portal for automated agreements analysis using NLP technologies</td>
</tr>
<tr>
<td>● Graphics and stats embedded on the website</td>
<td>● Visualization app featuring the network of EWC entries and actors</td>
</tr>
<tr>
<td>● Data dumps/queries for researchers or stakeholders to conduct analysis</td>
<td>● Portal for data linking to external data sources</td>
</tr>
<tr>
<td>● Connection to BI software such as Tableau</td>
<td>● Mobile applications for the EWC users</td>
</tr>
</tbody>
</table>

Table 2: Description of software applications as services

In general, the concept of services is flexible enough that these types of inventories can be extended. In a broader sense, we can think of future services, as long as the central principle that software applications are kept detached from the data, is preserved. Finally, the one and only gateway through which the structured data and the software applications will communicate is an API.

2.4. **Diagnosis Conclusion**

The main purpose of a knowledge database is to be consumed by users through services; therefore, in the long term, if it is decided for the data to be migrated to a content manager system, it would only result in a decrement of its capabilities. Nevertheless, if we take into account that the website and search engine are actually web services, then it’s possible to find a way to consider it as an independent module of the architecture, and ultimately connected to an overall data tasks manager that would do the weightlifting of the data access and protocols: an API.

A RESTful infrastructure combined with a CRUD interface for users is rather recommended. Regardless of the framework or technology, in the following part of the report, we will focus on particular technologies, although they are not exclusive. However, if it follows up on solutions like content managers, then we would be assuming that the only service is the webpage, which is not true. Once an API is implemented, the type of services or connection to applications are virtually limitless, scalable, efficient and more effective regarding its maintenance.
2.

The following section describes in detail the proposed paradigm for structuring the database previously discussed in the value proposition, but also indicated in the conclusion of the diagnosis.

2.1. **What is an API?**

An application programming interface or API is the internationally recognized standard for communicating, sharing and exposing data to services. An API is a computing interface that manages those interactions by defining the types of requests or calls that are possibly made, as well as the way to make them. Importantly, the API defines the data formats that should be used and the conventions that need to be followed, so that there is a homogeneous set of rules when interacting with data. It can therefore be seen as a “universal socket” that connects a data source to the external digital world.

A number of research communities have adopted the “API culture” by exposing their research data sources to the public or their own communities. There are a number of websites where one can find a registry of APIs along with their documentation platforms, such as Smart API platform. These registry platforms allow external users to get to know the existence of the data source, but not gain direct access to it; rather it presents the protocols of how to access the data complying with the FAIR principles for data management.

On the other hand, commercial oriented registries have become rapidly known among data providers, internet users and, in general, data science developers. A popular

---


8 https://smart-api.info/


API registry platform is Rapid API which not only provides an interface to query thousands of available APIs, (free and commercialized ones), it also provides templates for API documentation which is crucial when creating an API for data communication and making consumption more accessible.

The documentation of the API is the collection and description of all the HTTP endpoints that one will make available for web services. In general, there are several ways of documenting those endpoints, but it is best practice to create the API documentation with an embedded UI either on the main database reference website.

One example is the The World Bank Database, where one can easily find their API documentation and look for datasets of one’s interest.
In Figure 4 one can see that the structure of the endpoints depends on the type of data a user wishes to query, where, generally speaking, this would be defined by arguments in the endpoint. For instance, taking the following endpoint we could analyse the type of query that has been made:

![Figure 4: API documentation landing page of the World Bank database.](https://rapidapi.com/)

![Figure 4: API documentation landing page of the World Bank database.](https://www.worldbank.org/)

![Figure 4: API documentation landing page of the World Bank database.](https://datahelpdesk.worldbank.org/knowledgebase/articles/898581)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EWCdb layout Example</td>
<td><a href="http://www.etui.org">www.etui.org</a></td>
<td><a href="http://api.etui.eu/V1/(TABLE)?(ARGUMENT)=%7BVALUE%7D">http://api.etui.eu/V1/(TABLE)?(ARGUMENT)={VALUE}</a></td>
</tr>
</tbody>
</table>

Table 3: Example of a API url endpoint

2.2. **Why is an API suitable for EWCdb?**

One common mistake when implementing an API data access occurs when it is mapped and generated from the existing data model, like mirroring the data
model. However, we must bear in mind that an API call tell you what you can do with the database while the data model tells you what you have. In section 2.2 two types of users were defined, a contributor and a consumer, and these types of users will define the API actions.

A database consumer would perform one type of data transaction, which is querying
- an equivalent RESTful operator would be GET. A database curator would rather make additions and editing within the database - equivalent of RESTful operators would be POST, PUT and DELETE when necessary.

It is expected that these operations are performed by a consumer or contributor through a user interface, i.e., humans performing human-machine interactions; however, these are not the only interactions possible with an API. An API specially allows one to perform machine-machine interactions, i.e., the database communicates directly with applications in the same language.

An API as database controller:

- Can connect to a virtually limitless number of applications natively with no connectors or middleware, since the API environment serves as HTTP proxy itself.
- Allows an unlimited number of types of protocol access, and these protocols are designed ad-hoc, generally with a Token administration system where an unlimited set of types of users can be created.
- Connects to any web application, e.g. websites with dynamic content like media, search engines or dashboards.
- Can be "monetized". Normally, commercial databases charge for a certain level of protocol access using a unique Token. Furthermore, research and government-oriented databases use this model to make the users register and request a Token even though the content of the information is public, and this provides the administrator with the following information on data usage and provenance:
  - When was it used?
  - For which applications?
  - For how long the data was in use?
  - How many API calls were performed?
  - Was it performed on different IP addresses? Etc.
- Can be registered in API repositories and therefore comply almost natively with the FAIR principles for data management (Findable, Accessible, Interoperable and Reusable). The FAIR principles were first drafted at an international scientific workshop in 2014, with the principles officially being published in 2016. Since then, the principles have received worldwide recognition and are endorsed by international organisations including FORCE11, National Institutes of Health and the European Commission as an essential framework for sharing data and outputs in a way that will maximise use and reuse.
- Ensures sustainability of the systems. This is because an API is serving as a connector and does not directly depend on the rest of the systems; therefore the systems are
more resilient. For example, if the team decided to migrate from MySQL to Postgres, the API would be functioning normally.

- Holds the security aspect of the information, since the security protocols are now handled in the API level and not at the database level anymore.
- Can develop in parallel different branches or versions of the API, while serving without needing shutdowns or migrations.
- Can be implemented in a vast number of options and frameworks. This report particularly focuses on ExpressJS¹⁴ + SwaggerUI¹⁵, but we also have had experience creating APIs with Flask¹⁶ + Swagger. Furthermore, it is well known that content manager systems like Drupal 8¹⁷ can support API development. In fact, Drupal 8, natively performs RESTful API tasks for content administration in websites, however, sometimes the API definitions seem to be hardcoded and not very semantically valuable.

Most importantly, an API can be connected to a CRUD application, (explained in the following section). In a CRUD application, different users have the ability to access and manipulate the data. Modern CRUDs are developed in ReactJS¹⁸ or NodeJS¹⁹ frameworks. In other words, a CRUD application can be interpreted as the central GUI for the database using the API.

![API Gateway Manager](https://expressjs.com/)

Figure 5: API framework using ExpressJS and SwaggerUI

---

¹⁴ https://expressjs.com/
¹⁵ https://swagger.io/tools/swagger-ui/
¹⁶ https://flask.palletsprojects.com/
¹⁷ https://www.drupal.org/8
¹⁸ https://reactjs.org/
¹⁹ https://nodejs.org/

### 2.3. **What is a CRUD application?**

A CRUD application is a software that creates an interaction between a user and a database. The acronym CRUD stands for four basic functionality types: Create, Read, Update and Delete. The CRUD functions are inherent to databases consisting of different tables with rows and columns of data. The user can perform operations on the data either through code or a GUI. The Create function allows a new object to be added to the database with each new entry being assigned a
unique ID. The Read function allows all the objects in the database to be seen or searched without changing any of the stored information. The Update function enables stored information to be changed. This can involve modification of the existing information or the addition of certain fields (but not entirely new objects). The Delete function removes an object from the database. It is used to find a specific object, delete it and leave the other objects untouched.

As an example, consider the table with two rows below\(^\text{20}\). In reality the tables can contain thousands of rows and columns, but this provides a visualization of the CRUD app.

![Table Example](image.png)

**Figure 6: CRUD example implemented on a JavaScript based framework**

There are a number of frameworks with CRUD implementations, from classical PHP\(^\text{21}\) solutions, to content managements oriented using Drupal, but also there are more modern solutions using either JavaScript with React JS or Python with Django. In general, CRUD acts as a useful framework that reminds developers and users how to construct complete and usable data models.

\(^{20}\)https://github.com/nestjsx/crud
\(^{21}\)https://www.php.net/

### 2.4. The EWCDB Portal: a CRUD app implementation

In section 2.2, users were divided into two roles: contributor and consumer. These roles have technical equivalences with the systems with which they are communicating, i.e., a user consumer will perform Read operations whereas a user contributor will perform Create, Update and Delete operations.

<table>
<thead>
<tr>
<th>EWCdb user roles</th>
<th>CRUD</th>
<th>SQL</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>Read</td>
<td>SELECT</td>
<td>GET</td>
</tr>
<tr>
<td>Contributor</td>
<td>Create</td>
<td>INSERT</td>
<td>PUT</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>UPDATE</td>
<td>PUT</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
Table 4: User roles equivalences

MS access has been the system in current use as an alternative for a CRUD system, however MS access is not a CRUD system perse; therefore, a proper CRUD application should be allocated.

In the market of CRUD implementations there are endless solutions, frameworks and plug-and play systems ready for having a CRUD application running, however it is important to distinguish that there are two general paradigms for the deployment of these CRUD applications: **On premise, and Serverless.**

Having a CRUD application on premise would mean that there is dedicated space in the ETUI IT capacity, moreover the maintenance and CI/CD must also be hosted by the ETUI. On the other hand, creating a CRUD application and deploying it in a serverless manner will give the freedom of not having to deal with load balancing issues, request capacity, shutdowns, migrations, etc. Furthermore, a serverless CRUD app is an independent block on the overall architecture and the user interface can be continuously improved if it is a web-based application. Additionally, a friendly interface access to manage users’ rights and credentials can be included in the user interface.

This report recommends a state-of-the art serverless application framework to handle CRUD operations. Normally, an efficient solution would generate output from database contents (query results, automated statistics), including online applications (websites, social media). The ideal implementation for scalable CRUD is Amplify provided by Amazon Web Services. AWS Amplify is a set of tools that can be used together with other AWS tools or on their own, in order to help front-end web and mobile developers build scalable full stack applications in general, but particularly powerful CRUD applications. Using AWS Amplify one can configure web-based application backends and easily connect services, as well as deploy static web apps and manage together app content outside the AWS console.

![Figure 7: CRUD implementation framework using ReactJS hosted on AWS Amplify](image)

The main idea would be to design a user interface using ReactJS and deploy it to AWS via Amplify. Using Amazon Web Services would require a cost for the application hostage and deployment but compared with other cloud solutions and the time spent on self-engineered solutions, it is very reasonable. Here is an example of how the development and use of Amplify to host a CRUD
interface would cost ETUI around 16€ a month, even in the most expensive scenario. Given the usage it could be really be under 5€ a month of cost.

---

22 http://aws.amazon.com/
23 https://aws.amazon.com/amplify/
24 https://aws.amazon.com/amplify/pricing/

<table>
<thead>
<tr>
<th>Example 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A startup team with 5 developers have an app that has 300 daily active users. The team commits code 2 times per day.</td>
</tr>
<tr>
<td><strong>Monthly build &amp; deploy charges</strong></td>
</tr>
<tr>
<td>* Assumptions: Average build time = 5 mins; Number of work days/month = 20</td>
</tr>
<tr>
<td>* Total build time per month = num of devs * num of commits/day * num of days * avg build time = 5<em>2</em>20*3 = 600 build mins per month</td>
</tr>
<tr>
<td>* Monthly build &amp; deploy charges = 600*0.01 = $6</td>
</tr>
<tr>
<td><strong>Monthly hosting charges</strong></td>
</tr>
<tr>
<td>* Assumptions: Web app size = 25 MB, average size of page requested = 1.5 MB</td>
</tr>
<tr>
<td>* Monthly GB served = Daily active users * average page size * days = 300 * (1.5/1024) * 30 = 13.18 GB</td>
</tr>
<tr>
<td>* Monthly GB stored = web app size * number of monthly builds = (25/1024)(5<em>2</em>20) = 4.88 GB</td>
</tr>
<tr>
<td>* Monthly hosting charges = 13.18*$0.15 + 4.88*$0.023 = $1.97 + $0.11 = $2.08</td>
</tr>
<tr>
<td><strong>Total monthly charges</strong></td>
</tr>
<tr>
<td>Total charges = Build &amp; deploy charges + Hosting charges = $6+$2.08 = $8.08 per month</td>
</tr>
</tbody>
</table>

Figure 8: AWS Amplify low-cost billing example

Obviously, the CRUD application is a concept and not a technology per se; therefore, it must be understood that any other framework capable of creating CRUD applications is reliable for this task. It can be done with an open software framework as mentioned in section 3.3.

2.5. **Proposed Data Infrastructure**

A services-oriented data infrastructure cantered on a REST API is best fit for the EWCdb knowledge database. Figure 9 details the building blocks of the proposed EWCdb Architecture. The main component is the Data Access UI, mainly discussed in section 3.4. The RESTful API Environment is explained in section 3.2. The Data Services component is addressed in section 4 which is then divided in subsections giving examples of each service.
On the other hand, the Isolated Server Environment component is a step that it already exists in the overall picture. It is discussed in section 2.1, but it is important to keep in mind the concept of a “storage-only” SQL database. Virtually, users should not be able to "see" the database records, only administrators. Following this rationale, the security of the data tends to increase, and purity of the information provenance remains intact. In section 4.3 is explored how best communicate this operational database to Tableau Software where simple table views can act as proxy for a Datawarehouse.

2.6. Requirements

This report proposes a CRUD implementation framework using ReactJS that is connected to a RESTful API. In order to create a successful API, developers need to possess knowledge about API frameworks generally speaking, design and architecture, including the role of data, messaging and other services. They should have a thorough understanding of the REST architectural style and its use of HTTP. RESTful systems support different formats, such as HTML, YAML, XML and JSON, but JSON is recommended as it is a lightweight, easy way to exchange data and maintain its structure.
Furthermore, OAuth\(^{25}\), JWT\(^{26}\) and Webhooks have become popular technologies for making data and resources accessible, secure and easily integrated. The developers should be confident to explore new approaches that might be appropriate for the organisation.

The developers in charge of this should also possess professional communication skills in order to collaborate efficiently together with the researchers and users, as well as in order to liaise with other interested parties discussed in section 2.2. Consistent and precise documentation skills are essential to maintain the understandability and adaptability of the API for other developers and data consumers. Tools like Docusaurus\(^{27}\), Ram\(^{28}\), or Apiary\(^{29}\) are available to assist with documentation. However, Swagger\(^{30}\) is recommended in this case (section 3.2). Furthermore, the team in charge need to have a solid grasp of security (in order to prevent data breaches through hacked, exposed or broken APIs), cloud platforms (AWS is explored in this report) and testing tools, such as SmartBear\(^{31}\), API Fortress\(^{32}\), Rspec\(^{33}\), API Science\(^{34}\), Parasoft\(^{35}\), Postman\(^{36}\), or Runscope\(^{37}\) (in order to test endpoints and check responses).

\(^{25}\) https://oauth.net/
\(^{26}\) https://jwt.io/
\(^{27}\) https://docusaurus.io/
\(^{28}\) https://raml.org/
\(^{29}\) https://apiary.io/
\(^{30}\) https://swagger.io/
\(^{31}\) https://smartbear.com/
\(^{32}\) https://apifortress.com/
\(^{33}\) https://rspec.info/
\(^{34}\) https://www.apiscience.com/
\(^{35}\) https://www.parasoft.com/
\(^{36}\) https://www.postman.com/
\(^{37}\) https://www.runscope.com/

Regarding the technology stack, the main concerns for API development are frontend and backend frameworks: collections of libraries, languages and utilities used to build applications. The frontend framework includes the services used to create the user experience, including user functionality and the interface, and involves caching, wireframing, synchronization and UI development. The backend framework includes the database server, involving data storage and integration, server-side logic, versioning and push actions. These two frameworks are connected via the API. In the context of this report, the recommended backend framework is discussed in 3.2 and 3.4. where ExpressJS and AWS Amplify are connecting middleware, and the recommended frontend framework is ReactJS. ReactJS has a fast runtime, is lightweight, has
an extensive developer community available to provide assistance, and allows for multi-page and cross-platform applications.

The implementation time to create a successful API is ultimately dependent on the developers’ experience, skills and understanding of both the task and the technicalities. Frontend and backend infrastructures are each estimated to take about two to six months to be developed each (although they can be developed simultaneously by different developers), and then approximately two to three weeks should be budgeted for launching and testing.

2.7. Alternative Technologies

This report proposes the ReactJS stack; however, there are alternative options, the second recommendation being VueJS\(^\text{38}\), another frontend opensource framework with excellent performance. While React needs to be used in combination with other libraries, VueJS is standalone. VueJS is also fast and easy to learn, fully fledged but still lightweight, and has an increasing support base, an efficient documentation system, fast start-up times and better memory allocation than ReactJS. However, ReactJS is still more popular as it better allows for complex applications and has an enormous number of resources, documentation and a highly active community.

On the side to that, there are more modern solutions on which all the stack is integrated in one framework, Prisma.io\(^\text{39}\) is a great example of it. By simply defining the data model. Prisma would take care automatically with the migration and backend connection, on top of that, Prisma Studio provides an already build UI for editing the data

38 [https://vuejs.org/](https://vuejs.org/)
39 [https://www.prisma.io/](https://www.prisma.io/)

---

Figure 10: Prisma.io architecture

---

3. The main advantage of having an exposed API is that it is possible to natively connect to any web application.
4.1. **Connection to website and search engine**

The ETUI relies on a robust set of technologies to host its website and search interface for the EWCDb data. These technologies have been mastered and the implementations are mature enough to be sustainable in the long run with existing infrastructure.

The existence of a REST API that can connect to the search interface can only make things easier. It follows the same principles in terms of technology stack explored in section 3.4 where a frontend framework is chosen and a backend to host it.

Search interfaces can always be improved, and an important point remember is to record and save prior searchers from users so that researchers and database administrators can analyse the user journey looking for relevant legislation, litigation cases or agreements in the database. Having a log story of those searchers with the API would only make the search engine experience richer. The design of a search should mirror that of the rest of the data, simple
search panel after the results are displayed can be powerful since allows the user to re-try the search with different criteria without wasting time going to a new page or window. Here is where GET requests have some superior advantages over other solutions.

These concepts are very important not just from user and search engine, but from an architectural, API design point of view.

4.2. Connection to Visualization frameworks

Currently, the Graphs and Statistics are embedded in the websites using amCharts\(^\text{40}\) (see figure), which allows you to plug in predefined visualizations on the website. This approach is rather optimal; however, the maintenance of the visuals is manual as well as the data connection. Having an API would allow one to make streams of data requests to any visualization framework or dashboard, including amCharts, saving time and resources.

\(^{40}\)http://www.amcharts.com/

In addition to the existing tooling there are several “of-the-shelf” implementations for data visualization that can be fairly straightforwardly integrated into an existing REST API. In the following subsections D3.js and Plotly-Dash will be discussed.

Figure 13: amChart plug in in the EWCdb land page.
4.2.1. **Data Visualization with D3**

D3.js is a JavaScript library that is normally used for data visualization. D3 stands for data-driven documents, which describes the main paradigm of his technology: D3 manipulates documents based on data. D3 uses native HTML, SVG, and CSS which makes its web applications friendly by design. It does follow web standards; therefore, its web native and modern browsers can render these documents without tying to any proprietary framework.

The behaviour is very similar to that of amCharts: it is also JavaScript based software it is possible to create powerful dashboards. The main difference is that D3 feels more native for data scientists and engineers while the amCharts framework is more understandable for full-stack developers.

The main advantage is that the installation of the package and reading of the data is extremely simple, taking one line of code each once there is a JavaScript instance in the server that will host the application.

![D3.js snippet example](https://d3js.org/d3-collection.v1.min.js""></script>
d3.request(https://etui-sample.api.com/agreements).get(callback);

4.2.2. **Data Visualization with Plotly-Dash**

Plotly-Dash\(^1\) is a data visualization Python framework for building ready-to-use applications based on the web, and the main principle is that it can create customed user interfaces in pure Python. It's particularly suited for anyone who

---

\(^1\) As of writing this, the documentation is not publicly available, but it seems to be similar to the existing Plotly. Dash framework. For more information, visit [Plotly-Dash](https://plotly.com/dash/).
works and understands data in Python. This framework is written on top of Flask, Plotly.js, and React.js.

Figure 15: Plotly-Dash open-source gallery.

---

https://plotly.com/dash/

Plotly-Dash apps are rendered in any web browser. Through a couple of simple patterns, Plotly-Dash gets rid of all of the technologies and protocols that are required to build an interactive web-based application, which makes it simple enough so that a user (developer) can bind a user interface around a Python code in a really short amount of time. It is capable to deploy web apps to servers and then share them through URLs. Since Plotly-Dash apps are viewed in the web browser, these applications are by definition cross-platform and mobile ready.

Having this in mind, the only thing that is needed is a data handler piece of software, and this is where the main advantage of having a REST API comes into play. The REST API can communicate directly via Python drivers to the Plotly-Dash application, i.e. it calls for the data sources from the Python code.

The open-source Python library “requests” together with “JSON” are enough to integrate the type of REST call GET and access the data in a fairly easy manner. Within somelines of code one can access a data instance of the EWCdb, compelled with the required credentials and it can be parsed as JSON format. The JSON data format is a type of data object which Python programming language can understand.

Figure 16: Example of API communication with Python for data handling.

```python
# Python 3.7
import requests
import json

url = 'https://etui-sample.api.com/agreements'
response = requests.get(url, auth=('user', 'pass'))
data = response.json()
```

Once the data object is defined it can be passed directly to the components included on the Plotly-Dash library and ultimately apply the visualization functions defined.

```python
# Python 3.7
import dash_core_components as dcc
import plotly.express as px

fig = px.scatter(data, x, y)
dcc.Graph(figure=fig)
```
Using Plotly-Dash for creating proof of concept web applications is a preferred solution in the Data Science community, since is easy to implement, deploy and maintain. It does not require extra storage capacities and the only requirement is to know Python. The great disadvantage is that is not meant to be a long-term solution for production-level products.

4.3. **Connecting Tableau**

Tableau\(^{42}\) is a Business Intelligence (BI) software for conducting classical statistical analysis, data analytics, and dashboards creation. Currently there are two options for connecting it: 1) via third party REST API driver connection or 2) By creating tailored *table views* and connecting directly to the database.

4.3.1. **Options and bridges**

The second is recommended, given that third party connection would run into assistance and support issues and it is precisely one of the things that one prefers to avoid. The only point to take into account is that some time is required to be invested to create research relevant database *table views*.

![Database connection to Tableau](image)

To connect to the Database one needs: The name of the server that hosts the database you want to connect to, the username and the password. One also needs to have one of the Tableau services installed in the user’s PC.

There are different ways in which ETUI can benefit from Tableau. Currently Tableau Desktop is the option for which licences were acquired.

---

\(^{42}\)https://www.tableau.com/

<table>
<thead>
<tr>
<th>Tableau Desktop</th>
<th>Made for individual use in a PC, the licence number can’t be shared</th>
</tr>
</thead>
</table>
Tableau Server | It has the same functionalities that Tableau Desktop with the difference that is expected that the organization has a huge number of users so that they can collaborate on the reporting aspect

Tableau Online | Business Intelligence in the Cloud. In essence it has the same capabilities than Tableau Server, with the advantage that everything would be on the cloud and does not depend to any ICT skillset

Tableau Reader | This is a free product, that is recommended to be downloaded by anyone in the team. Tableau Reader allows you to open files saved in Tableau Desktop on read-only mode.

Tableau Public | This is also a free service. Anyone can open a Tableau Public and use the same Tableau Desktop installed to create dashboards. The difference is that with Tableau Public is required to save all changes in the Tableau Public webpage, which is open to all. Public, therefore is not recommended for sensitive content or not open data.\(^43\)

Table 5: Applications for creating data visualization solutions with Tableau

Once a Tableau application is installed in the local machine, a driver is required. This driver makes the bridge between the Tableau application and the database. Normally, these type of Database drivers are not installed in PCs by default. If the driver is not installed on the users’ computer, Tableau will display a message in the connection dialog box with a link to the driver download page where you can find driver links and installation instructions.\(^44\)

\(^43\) [https://public.tableau.com/gallery/](https://public.tableau.com/gallery/)

\(^44\) [https://www.tableau.com/support/drivers](https://www.tableau.com/support/drivers)

In the following example, the selected Database type is MySQL and the operating system is Mac.

**Figure 19: An ODBC driver is always required to bridge Tableau and the Database in the Desktop versions**
4.3.2. **Tableau special features**

- **Tableau Mobile**

Among the different services provided by Tableau, the mobile version is very interesting since it can connect to dashboards published in public URLs or via the Tableau server account. The functionalities are similar to the Tableau reader. What is important is that creating a layout for each device type gives the most control over your users’ experience as they interact with the dashboards from different devices. After a dashboard is published with all three layouts, users will not see the default dashboard layout; instead, they will always see the appropriate device-specific layout.

- **Tableau Extracts**

It is important to acknowledge that even the most beautiful dashboard will not have an impact if it takes too long to load. Normally, long load times are caused by the data complexity or the size of the workbook, or a combination of the two.

Therefore, there are some critical decisions that must be made as an author in the data preparation stage, especially about production views. It is recommended to perform calculations in the database to reduce overhead. Furthermore, precomputed aggregations are great for calculated fields in Tableau. Most importantly when a dashboard is published it is best practice that it is accompanied by a *Tableau Data Extract*.

![Figure 20: Tableau extracts abstraction](image)
A Tableau data extract is a permanent copy of the data used for specific views of a dashboard. This option can be easily activated from the user interface. The objective is to reduce the load time of data querying to the database every time a spreadsheet reads the data, therefore optimizing loading time. Considering filters on a data source and creating extracts are typically much faster ways to deal with live data. Nevertheless, data extracts are not a long-term solution when querying against constantly refreshing data, and a live connection often makes more sense for operationalizing the view from the database level.

- Tableau Set Actions

Set actions is a feature that aims to work with a selection of data points dynamically in a visualization task and store it in a set. Set actions extend the interactivity to support custom user-defined behaviour instead of hard-coded selections.

Figure 21: Tableau Set Actions abstraction

In principle, a regular set can apply different behaviours to various target sheets. For example, the same set could colour viz A, hide data in viz B, and filter an axis in viz C. By using set actions, a user selection in any of the sheets can update the set, thereby modifying all target sheets in a single coordinated selection. Coordinating multiple actions through a single selection dramatically increases the breadth and depth of scenarios that can be addressed for end users through interactive analytic applications.
4.4. **Data Analytics & Automated analysis of agreements**

Today multiple computational tools are already being deployed in legal fields and policy making, such as data mining, machine learning, deep learning simulations, natural language techniques, document management, legal text analytics, computational game theory or network analysis. These tools capture rich and detailed data about the external world, make them computable, and process them to reach a broader and more granular level of analysis. For instance, it is possible to use technologies like OpenAI and use the GPT-3 models for high scalable NLP tasks, such as text summarization, structuring unstructured text, case law facts analysis, text generation, etc.

Figure 22: Example of GPT-3 output of AI-generated text using OpenAI API

It is possible to analyse merger control and agreements content from a text mining point of view, where techniques such as named entity recognition could be utilized to detect the main actors and patterns in the text. On the other hand, there is important research on policy document sentiment analysis, where one can create scores of urgency, riskiness or positiveness in a policy document. These techniques have been also applied to case law.

4. **Final Remarks**

It has been a pleasure to collaborate with ETUI on this report and to generate applicable and specific recommendations that aim to improve the functioning and security of ETUI’s operations and information. The consultant and the ETUI share the right to use all intellectual results generated during this
project and will coordinate such use between them. The consultant has ensured adequate data protection and will not disclose any information regarding ETUI’s information with any other party. No person’s individual data has been used for this research; thus, no GDPR or other legal obligations regarding personal privacy and security have been involved. The technologies mentioned are recommendations provided by the consultant, who has no commercial association with any of the products mentioned, no intellectual ownership over the technologies and no alternative agenda regarding his recommendations other than to assist the ETUI with its objectives. Lastly, in the event that the researcher does not implement the suggested architecture, any problems arising with regards to the implementation and functioning are not his responsibility in any way.

47 Similarity and Relevance of Court Decisions: a Computational Study on CJEU Cases
http://ebooks.iospress.nl/volumearticle/53654