Nghi Thanh Le

HYBRID DATABASE FOR XML RESOURCE MANAGEMENT

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Prof. Kari Systä
Niko Siltala
April 2023

ABSTRACT

Nghi Thanh Le: Hybrid Database for XML Resource Management
M. Sc. thesis
Tampere University
Master’s degree program in Computing Sciences
April 2023

Although XML has been used in software applications for a considerable amount of time, managing XML files is not a common skill in the realm of backend software design. This is primarily because JSON has become a more prevalent file format and is supported by numerous SQL and NoSQL databases. In this thesis, we will delve into the fundamentals and implementation of a web application that utilizes a hybrid database, with the goal of determining whether it is suitable for managing XML resources.

Upon closer examination of the existing architecture, the client discovered a problem with upgrading their project. Further investigation revealed that the current approach of storing XML files in a single folder had serious flaws that could cause issues. As a result, a decision was made to revamp the entire web application, with hybrid databases being chosen as the preferred solution due to the application's XML storage concept. It is worth noting that there exists a type of database specifically designed for XML resources, known as native XML databases. However, the development team thoroughly reviewed all the requirements provided by the product owner, Niko Siltala, and assessed the compatibility of both native XML databases and hybrid databases for the new application. Based on our analysis, it was concluded that the hybrid database is the most suitable option for the project.

The changes were successfully designed and implemented, and the development team determined that hybrid databases are a viable option for managing a significant number of XML file dependencies. There were no significant obstacles encountered that would hinder the use of this type of database. The advantages of using hybrid databases were observed, including streamlined XML file storage, the ability to mix XPATH/XQUERY in SQL queries, and simplified codebases.

Keywords: XML, resource management, hybrid database, native XML database.

ACKNOWLEDGEMENT

As someone new to the concept of managing XML resources in web applications, I chose to focus on this topic for my master's thesis. The project was completed in under a year, with the thesis writing process taking approximately six months. The idea for this thesis originated from a project that I worked on as part of my job, which I found interesting since I was not previously familiar with databases that support storing XML files. Throughout the project, I had the freedom to design the architecture as I saw fit, while also receiving support and guidance from my co-workers, especially Niko Siltala, who was patient in explaining the original project's concept and the importance of XML files to his platform.

I am grateful to Tampere University, my employer, and EU Horizon 2020 ODIN project (grant agreement no. 101017141), for providing me with the opportunity to write my master's thesis on this topic. I also extend my appreciation to product owner Niko Siltala for allowing me to use their project as the subject of my research. Finally, I would like to show appreciation to Kari Systä and Niko Siltala, who were my thesis advisors, for their insightful criticism and encouragement throughout the writing process.

Tampere, 21 March 2023

Nghi Thanh Le

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

HTML	HyperText markup language
XML	Extensible markup language
PDF	Portable document format
XSLT	Extensible stylesheet language transformation
REST	Representation state transfer
API	Application programming interface
JSP	Java server page
RD	Resource description
W3C	Word wide web consortium
HTTP	Hypertext transfer protocol
cURL	Client for URL
SQL	Structured query language
BLOB	Binary large object
CLOB	Character large object
JDBC	Java Database Connectivity
MSSQL	Microsoft SQL Server
SOAP	Simple Object Access Protocol
RPC	Remote Procedure Call
ODIN	Open-Digital-Industrial and Networking

LIST OF TABLES

1. INTRODUCTION

Markup languages are used to format and document the contents of an entity, product definition, and a web page. Common markup languages include HTML, XML, and PDF. Among the list, XML is one of the most popular choices to store data because data stored in XML is readable, understandable and an option of transferring data [13]. In addition, it is scalable and extensible which helps it to be more compatible with distinct types of platforms and devices. Based on those advantages, XML is a maturing technology for the organization of complex data structures like robotic information for Java applications and building RESTful APIs for modern apps. Broadly speaking, XML resources management can be seen as one of the core areas for such types of software applications consuming huge volumes of XML files from multiple sources and providing it to structured and semi-structured data for consumption by various applications.

As XML is a structured way to store data, however, XML is not a database, and it was never meant to be a database [1]. There are several approaches to establishing XML data storage. It is possible to save XML files in relational or non-relational databases via a special type of binary data. On the other hand, developers may store XML files at a specific location and use the database to remember the location path to frequently retrieve the data from these files. This is inefficient and requires a lot of storage space and regular maintenance. Software development should always aim for the best possible result by creating outcomes, products or services that create real value for the end users.

Since XML files play a critical role in providing content to users, for instance, transforming XML files into HTML by XSLT or into several other presentation formats. The management of these XML files becomes critical to achieve optimal performance and quality across all platforms. To overcome this limitation, hybrid databases were developed, which enabled developers to store the XML files in an efficient manner.

1.1 Background

Niko Siltala, the product owner, first implemented the Resource Catalogue Platform in 2008 based on a common java web application approach using Java Server Page (JSP) and Apache Tomcat. The project provides a resource management program allowing the users to store product information files in a folder. The file location and part of its information are saved in Maria database. It also has a user creation flow and a basic authentication method. The handling of

XML files is one of the major functions of the application. A user can upload a resource file to view human-readable information or have the overview of XML files stored in the folder. All XML documents were provided by developers and often needed to run a procedure to map XML files into the database entities. [2]

In the previous version, making changes to XML resources such as adding, deleting, or updating them was carried out by developers manually, meaning that they had to modify the XML code directly. This approach was not automated and required a certain level of technical expertise and attention to detail from the developer. The application itself lacks the ability to manage these interactions and would have to rely on manual services that could be unreliable or unavailable. This introduces risk and decreases the overall efficiency of the system. The product owner wanted to update the original versions to manage XML files more effectively. There are potential applications that can manage XML resources and provide much more flexibility in XML resource interactions.

1.2 Research question and research scope

To determine whether there is a more effective way to store XML data than the conventional method of saving it in a folder and creating an ecosystem around it, various kinds of XML databases were used in this research. The aim was to explore if a database management system could provide a more streamlined, automated, and scalable solution for XML storage and retrieval than the file-based approach. The study sought to evaluate the feasibility and potential benefits of using a database system for managing XML data in terms of performance, scalability, and ease of use. The research was conducted by utilizing the Action research strategy and was limited to the Resource Catalogue program. The development team consisted of myself and Niko Siltala. The project ended at the end of 2022.

The analysis of the legacy project led to consolidated assumptions about platform owner problems. Through the analysis of the data collected via questionnaires and the interviews conducted, the second objective was to devise a clear set of research questions to be answered to further understand the perceived problems and propose suitable solutions to address them. The more specific content of the research emerged during the analysis round and finally formulated the research question:

- What is the alternative database product to replace file folder storage for a better resource management platform?

1.3 Objectives

The present thesis aims to analyze the existing technologies and determine how they can be leveraged to solve the problems associated with managing large collections of XML files. In the thesis, the possibility of using a hybrid database to manage XML data is researched. The research was conducted by utilizing the Action research strategy but being limited to open-source or community licenses.

The first part of the project focused on understanding the product owner's requirements, which were the foundation of selecting the right software products. During the requirement elicitation phase, the development team established the objectives for the new application, which include accessing resources, querying resources, and browsing interface standard specifications. The second part of the project focused on rebuilding the original version and adapting the revamp into the selected tool. After we conducted the requirements for the new project, the second phase may start. The implementation includes finding suitable tools, implementing a new Java application, and configuring and customizing Application Programming Interface (API) documentation to generate the previous phase’s identified components. Along with the implementation period, we also resolved problems discovered in the earlier phases.

At the end of the research, the product owner reviewed the implementation and validated each feature based on the API endpoint specifications to determine the suitability of the chosen XML resource management application for the project.

1.4 Methodology

This work consisted of four different phases. The first phase involved establishing the state of the art regarding the different concepts related to XML, XML resource management, with a focus on various types of XML databases.

The second phase focused on requirements elicitation. We interviewed Niko Siltala, identified end-users, utilized documented software specifications, and designed application API endpoints to identify the different needs for ResourceCatalogueV2.

The third phase involved studying different XML databases to select the most suitable one for the application. A comparison between various tools was conducted to determine which ones best met the requirements identified in the previous phase.

In final phase, the prototype was designed and implemented. We used the action research methodology to conduct the results of this phase. Niko evaluated the system's efficiency based on his API documentation and software specifications for testing purposes.

1.5 Structure of the thesis

This thesis comprises of seven sections, each presenting distinct topics. The introductory section is followed by Section 2, which provides a comprehensive overview of resource management, XML technologies, and various XML databases. Section 3 outlines the original application design, followed by Section 4, which details the requirements elicitation process. Section 5 illustrates the proposed hybrid database solution and compares it with existing alternatives, along with documentation for the revamped version built on the chosen database. Section 6 presents the results of the new design and assesses if it fulfills the requirements. Finally, Section 7 concludes the paper.

2. THEORETICAL BACKGROUND

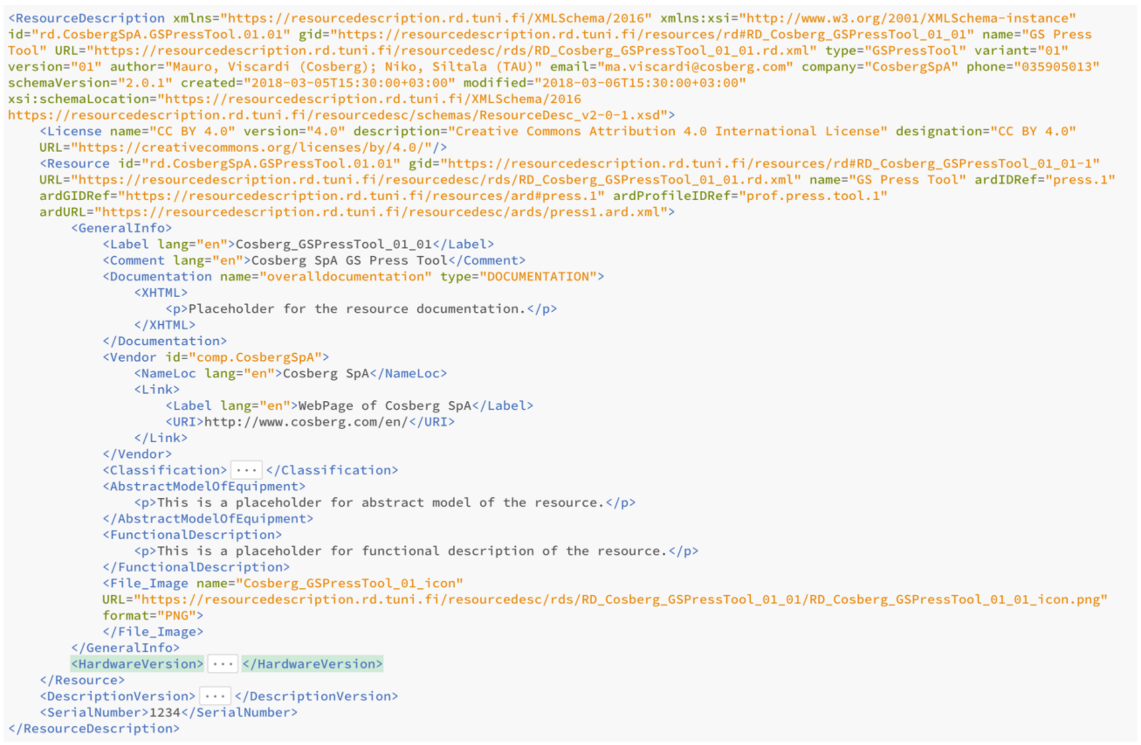
XML has gained popularity as web programming has progressed. In contrast to computer systems and databases that store data in various formats, data can be saved in XML in plain text format, making it independent of software and hardware. XML is also used to streamline the storing and sharing of data [3]. As a result, it becomes considerably simpler to create data that can be shared by numerous apps.

The aim is to optimize XML file management so that documents are safe, easy to integrate with software applications and as comfortable as possible. It is possible to improve the XML datastore by adding accessories. XML is already equipped with a wide range of accessories, and more are continuously being developed.

This chapter delves into the fundamentals of XML technology, its underlying principles, and available options for XML resource management. The first subchapter presents an introduction to resource management, the second subchapter examines XML technology, and the third subchapter discusses various approaches for storing XML files. Following the introduction of each paradigm, the chapter addresses challenges identified within each paradigm in the respective subchapters.

2.1 What is the resource under management?

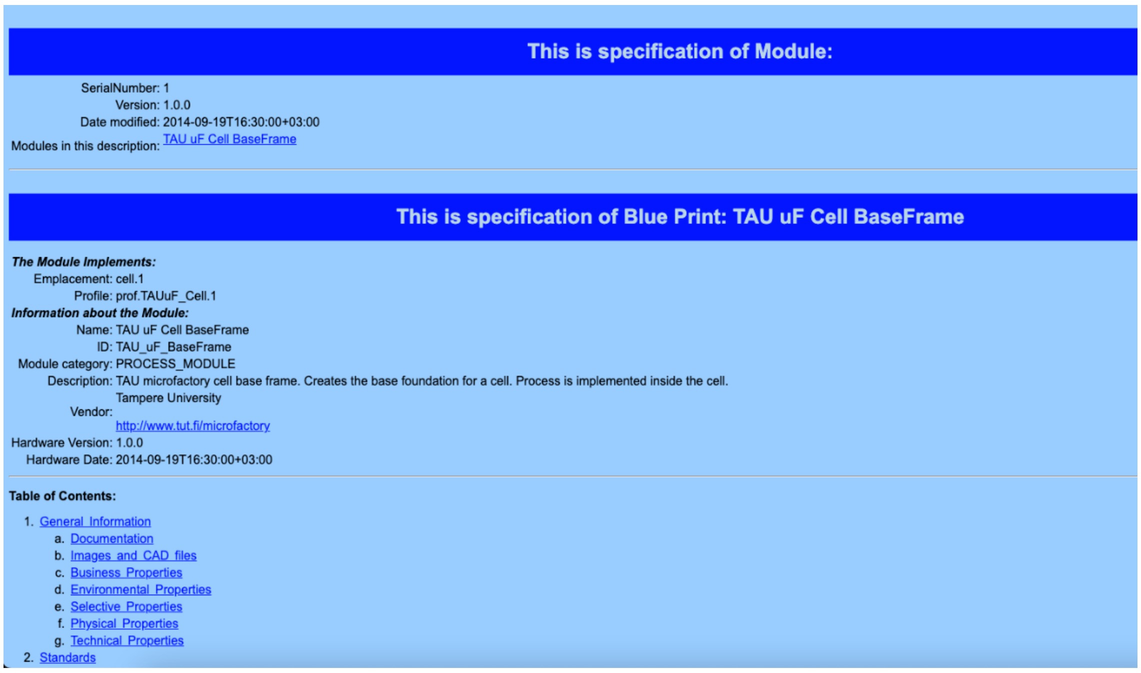
The Resource Description (RD) is a comprehensive representation of a technical entity such as balancer device, manipulator, or gripper that encompasses all aspects of its mechanics, and functionality. It gives a thorough explanation of the module's features, including how it works on the product and process sides and how it interacts with other modules. [4]



```xml
<ResourceDescription xmlns="https://resourcedescription.rd.tuni.fi/XMLSchema/2016" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="rd.CosbergSpA.GSPressTool.01.01" gid="https://resourcedescription.rd.tuni.fi/resources/rd#RD_Cosberg_GSPressTool_01_01" name="GS Press
Tool" URL="https://resourcedescription.rd.tuni.fi/resourcedesc/rds/RD_Cosberg_GSPressTool_01_01.rd.xml" type="GSPressTool" variant="01"
version="01" author="Mauro, Viscardi (Cosberg); Niko, Siltala (TAU)" email="ma.viscardi@cosberg.com" company="CosbergSpA" phone="035905013"
schemaVersion="2.0.1" created="2018-03-05T15:30:00+03:00" modified="2018-03-06T15:30:00+03:00"
xsi:schemaLocation="https://resourcedescription.rd.tuni.fi/XMLSchema/2016
https://resourcedescription.rd.tuni.fi/resourcedesc/schemas/ResourceDesc_v2-0-1.xsd">
    <License name="CC BY 4.0" version="4.0" description="Creative Commons Attribution 4.0 International License" designation="CC BY 4.0"
    URL="https://creativecommons.org/licenses/by/4.0/"/>
    <Resource id="rd.CosbergSpA.GSPressTool.01.01" gid="https://resourcedescription.rd.tuni.fi/resources/rd#RD_Cosberg_GSPressTool_01_01-1"
    URL="https://resourcedescription.rd.tuni.fi/resourcedesc/rds/RD_Cosberg_GSPressTool_01_01.rd.xml" name="GS Press Tool" ardIDRef="press.1"
    ardGIDRef="https://resourcedescription.rd.tuni.fi/resources/ard#press.1" ardProfileIDRef="prof.press.tool.1"
    ardURL="https://resourcedescription.rd.tuni.fi/resourcedesc/ards/press1.ard.xml">
        <GeneralInfo>
            <Label lang="en">Cosberg_GSPressTool_01_01</Label>
            <Comment lang="en">Cosberg SpA GS Press Tool</Comment>
            <Documentation name="overalldocumentation" type="DOCUMENTATION">
                <XHTML>
                    <p>Placeholder for the resource documentation.</p>
                </XHTML>
            </Documentation>
            <Vendor id="comp.CosbergSpA">
                <NameLoc lang="en">Cosberg SpA</NameLoc>
                <Link>
                    <Label lang="en">WebPage of Cosberg SpA</Label>
                    <URI>http://www.cosberg.com/en/</URI>
                </Link>
            </Vendor>
            <Classification> ··· </Classification>
            <AbstractModelOfEquipment>
                <p>This is a placeholder for abstract model of the resource.</p>
            </AbstractModelOfEquipment>
            <FunctionalDescription>
                <p>This is a placeholder for functional description of the resource.</p>
            </FunctionalDescription>
            <File_Image name="Cosberg_GSPressTool_01_icon"
            URL="https://resourcedescription.rd.tuni.fi/resourcedesc/rds/RD_Cosberg_GSPressTool_01_01/RD_Cosberg_GSPressTool_01_01_icon.png"
            format="PNG">
            </File_Image>
        </GeneralInfo>
        <HardwareVersion> ··· </HardwareVersion>
    </Resource>
    <DescriptionVersion> ··· </DescriptionVersion>
    <SerialNumber>1234</SerialNumber>
</ResourceDescription>
```

Figure 1. *A minimal XML implementation of capabilities and skills extract*

All files are formatted in XML and standardized using XML Schemas. The use of XML as the foundation for the Resource Catalogue Platform was deemed appropriate due to its advantages in terms of interoperability and interchangeability. The widespread availability of tools and libraries that support XML processing and utilization makes it a suitable common ground for information exchange [5]. ResourceCatalogueV1 processes and makes use of information that is accurate and comprehensive thanks to built-in mechanisms for XML file content verification. Compared to alternative solutions that employ a unique description language or are implemented in a particular programming language, XML technologies have a higher abstraction level and are more independent of tool and operating system implementations [5]. The Resource Description files are stored in a central location and can be accessed and viewed in multiple formats, such as human-readable documentation for RD files that can be generated on-demand. [4]


This is specification of Module:

SerialNumber: 1
Version: 1.0.0
Date modified: 2014-09-19T16:30:00+03:00
Modules in this description: TAU uF Cell BaseFrame

This is specification of Blue Print: TAU uF Cell BaseFrame

The Module Implements:
Emplacement: cell.1
Profile: prof.TAUuF_Cell.1
Information about the Module:
Name: TAU uF Cell BaseFrame
ID: TAU_uF_BaseFrame
Module category: PROCESS_MODULE
Description: TAU microfactory cell base frame. Creates the base foundation for a cell. Process is implemented inside the cell.
Vendor: Tampere University http://www.tut.fi/microfactory
Hardware Version: 1.0.0
Hardware Date: 2014-09-19T16:30:00+03:00

Table of Contents:

1. General Information
   a. Documentation
   b. Images and CAD files
   c. Business Properties
   d. Environmental Properties
   e. Selective Properties
   f. Physical Properties
   g. Technical Properties
2. Standards

Figure 2. *Human readable documentation of a RD file [2]*

As part of the Resource Catalogue Platform, RD should be published, searched, and viewed. RD files serve as exchange formats between different production system engineering tools, facilitating the reconfiguration and adaptation of production systems. [4]

2.2 Resource management in software development

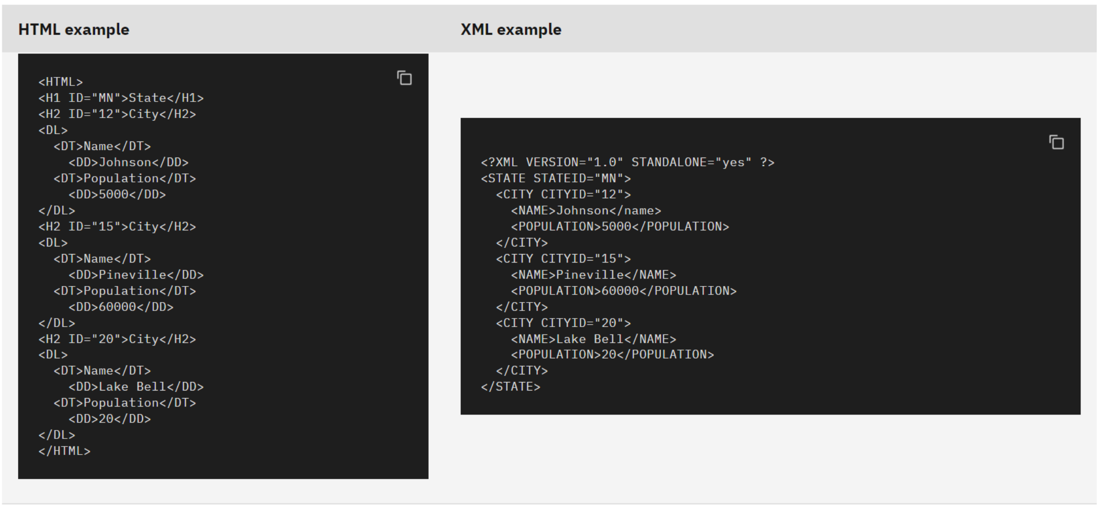
In software development, resource management refers to the efficient and effective allocation and utilization of resources, such as time, hardware, and software [6]. To ensure that a software application performs optimally and meets the needs of its users, proper resource management is essential.

Resource pooling, resource scheduling, and resource optimization are some of the strategies and techniques that can be used for managing resources in software applications. Resource pooling involves creating a central repository or pool of resources that can be shared among multiple users or processes. Resource scheduling involves allocating resources in a way that ensures that they are used efficiently and effectively. Resource optimization involves fine-tuning the use of resources to maximize their utilization and minimize waste [8].

An effective resource management strategy is important for a variety of reasons. By minimizing the need for additional resources or optimizing the use of existing resources, it can reduce costs and improve performance by ensuring that resources are used effectively and efficiently. In addition, valuable resource management can contribute to improved user experience by ensuring that the software application is responsive and reliable [6][7]. As a result, resource management plays a crucial role in software development and maintenance. By carefully planning and managing the use of resources, software developers and system administrators can ensure that their applications will meet the needs of their users and operate at optimal efficiency and performance levels [6].

2.3 XML definition

The World Wide Web Consortium (W3C) has approved the Extensible Markup Language (XML) as a standard for document markup in the web community. The XML specification is created and updated by W3C. In 1998, the first W3C recommendation was released [9]. It specifies a general syntax for data marking up and gives computer documents a standard format that may be altered to meet a variety of applications, including websites, electronic data exchange, object serialization, remote procedure calls, voice mail systems, and more [10]. Data are contained in XML documents as text strings. The data is surrounded by text markup that describes it. The fundamental unit of data and markup, known as an element, has an explicit syntax that is defined by the XML specification. This markup must follow several rules regarding element delimitation by tags, the appearance of tags, the acceptable names for elements, and where attributes should be placed. While XML markup appears like HTML markup on the surface, there are a few significant differences such as syntax flexibility, strictness, validation, semantic meaning [10]. The readability and extensibility of XML are demonstrated in the following example in a simplified manner.



HTML example

```
<HTML>
<H1 ID="MN">State</H1>
<H2 ID="12">City</H2>
<DL>
  <DT>Name</DT>
    <DD>Johnson</DD>
  <DT>Population</DT>
    <DD>5000</DD>
</DL>
<H2 ID="15">City</H2>
<DL>
  <DT>Name</DT>
    <DD>Pineville</DD>
  <DT>Population</DT>
    <DD>60000</DD>
</DL>
<H2 ID="20">City</H2>
<DL>
  <DT>Name</DT>
    <DD>Lake Bell</DD>
  <DT>Population</DT>
    <DD>20</DD>
</DL>
</HTML>
```

XML example

```
<?XML VERSION="1.0" STANDALONE="yes" ?>
<STATE STATEID="MN">
  <CITY CITYID="12">
    <NAME>Johnson</name>
    <POPULATION>5000</POPULATION>
  </CITY>
  <CITY CITYID="15">
    <NAME>Pineville</NAME>
    <POPULATION>60000</POPULATION>
  </CITY>
  <CITY CITYID="20">
    <NAME>Lake Bell</NAME>
    <POPULATION>20</POPULATION>
  </CITY>
</STATE>
```

Figure 3. *Comparing XML to HTML as an example [11]*

There are many uses for XML, including Web, e-business, and portable applications. Some of the applications include [12][13][14]:

- Data transfer: The ability to convey data in a structure that can be understood by these various programs makes this format a popular choice for building APIs. There are two types of XML APIs used in web services: Simple Object Access Protocol (SOAP) and Remote Procedure Call using XML (XML-RPC). Applications using either of these APIs only need to know their format to access the data it contains.
- Creating layouts: Android mobile applications use XML to create all layouts, which determine where data will appear on the phone's screen. The Linear Layout, which indicates whether the content should be aligned horizontally or vertically on the screen, the Frame Layout, which includes other layouts dynamically, and the List Layout, which displays items that can be scrolled through.
- Storing configuration data: The XML format represents low-level data, such as configuration files. This approach considers XML as a replacement for more traditional methods, such as Windows' INI files or Java's Property files, which use lists of name/value pairs.

These examples of XML in the industry highlight its core role. XML was conceived as a solution to intercommunication. Data is exchanged between components more easily, thereby allowing developers to concentrate on the more important aspects of coding, such as business logic, instead of constantly worrying about various formats of input and output. As a solution to the question of whether files should be easily readable by humans or by software, XML aims to satisfy both needs [13].

2.3.1 XML syntax

An XML document's elements, which are represented by tags, are its fundamental building blocks. An opening tag, content, and closing tag make up an element. The element name appears in the opening and closing tags and is enclosed in angle brackets [10]. Furthermore, elements can have attributes that reveal more details about the element. The opening tag contains attributes that are written in the format name="value". For example, the following XML code defines a DataTypes element with a StringType element inside it.

```
<Datatypes>
      <StringType id="dt.STRING.TODO_DEFINE" arrayLength="128"
name="STRING"/>
</Datatypes>
```

A root element that includes all other elements is a requirement for XML documents. All other elements in the document are contained within the root element, which is specified by a single pair of opening and closing tags [10]. Elements in XML may contain either text or other elements.

This nesting of elements within elements is utilized to express the hierarchical relationships between the various elements within the document. This arrangement of elements provides a means to describe the structure and organization of the data being stored within the XML document [15]. For example, the following XML code defines a root element called Inputs that contains multiple SymbolRef elements:

```
<Inputs>
   <SymbolRef idRef="var.acceleration" reqOpt="REQ" connectionMethod="ACY-
CLIC"/>
   <SymbolRef idRef="var.axisID" reqOpt="REQ" connectionMethod="ACYCLIC"/>
   <SymbolRef idRef="var.deceleration" reqOpt="REQ" connectionMethod="ACY-
CLIC"/>
   <SymbolRef idRef="var.position" reqOpt="REQ" connectionMethod="ACYCLIC"/>
</Inputs>
```

Program 1. *An example of XML elements with a root element*

In addition to elements and attributes, XML documents can also include comments, which are used to add notes or explanations to the code. Comments are written in the form <!-- comment --> and are ignored by the XML processor [10]. For example, the following XML code includes a comment explaining the schematype of the Resource element:

```
<ResourceDescription>
    <!-- https://resourcedescription.rd.tuni.fi/resourcedesc/schemas/Re-
sourceDesc_v2-0-1.xsd ../schemas/ResourceDesc_v2-0-1.xsd -->
    <License name="CC BY 4.0" version="4.0" description="Creative Commons At-
tribution 4.0 International License" designation="CC BY 4.0"
URL="https://creativecommons.org/licenses/by/4.0/"/>
</ResourceDescription>
```

Program 2. *XML code example with comments*

2.3.2 Well-formed XML documents

An XML document must follow specific syntax rules to be processed and understood by an XML processor. A document that meets these W3C's XML Recommendation is said to be well-formed [10].

To be considered well-formed, an XML document must have a single root element that encloses all other elements in the document. This root element helps to define the structure and hierarchy of the data contained in the document [10]. For example, consider the following XML document:

```
<ResourceDescription>
    <DescriptionVersion>
        <VersionNumber>1.0.0</VersionNumber>
        <DateTime>2016-10-13T16:30:00</DateTime>
    </DescriptionVersion>
    <SerialNumber>1212</SerialNumber>
</ResourceDescription>
```

Program 3. *A good example of an XML document*

In this document, the root element is ResourceDescription, and it encloses DescriptionVersion and SerialNumber elements. In the DescriptionVersion element, in turn, encloses two child elements: VersionNumber and DateTime. The hierarchy of the document's data helps to determine its structure.

A well-formed XML document must contain correctly nested elements in addition to a single root element. This means that there must be a closing tag to go along with every opening tag [15]. Proper attribute syntax is another condition of a well-formed XML document. The element's opening tag contains attributes, which are used to provide further details about an element [10][15]. They are written in the form name="value".

A well-formed XML document must also use the appropriate entity syntax. This necessitates replacing any special characters, such as, >, and &, with their respective entity codes. (e.g., &lt;, &gt;, and &amp;) [15]. In conclusion, a well-formed XML document is one that adheres to the fundamental XML syntax principles. It doesn't contain any syntax errors or invalid characters, has a single root element, appropriately nested elements, proper attribute syntax, and suitable entity syntax [10]. Below, we will discuss the solution to keep XML files in greater detail.

2.4 XML resources management

ResourceCatalogueV1 is an example of an application that contains a substantial amount of XML documents. As the application scales and the number of XML files grow, questions about how to effectively save XML files have also been considered. By looking at the way in which XML files are stored on a machine's file system, or how they appear as a data type in a row in a database application, we will be able to understand the concept of storing XML files. Storage of XML documents in file systems, stuffing XML data into large objects (LOBs) in relational databases, shredding XML data into numerous relational columns and tables, and isolation of data into XML-only databases are popular ways used to handle XML data. [16]

2.4.1 Native XML database

The Native XML database is one way of storing XML documents and data efficiently. Unlike relational databases, Native XML databases are built using a document data architecture pattern rather than tables. Neither a middle-tier object-relational mapping nor join procedures are needed by a document-oriented database to store or retrieve complicated data [17]. This kind of database is well-illustrated by the open-source database eXist. The source code and binary for eXist, which was created in Java, are both freely accessible [19]. The following figure displays an overview of the eXist system architecture.

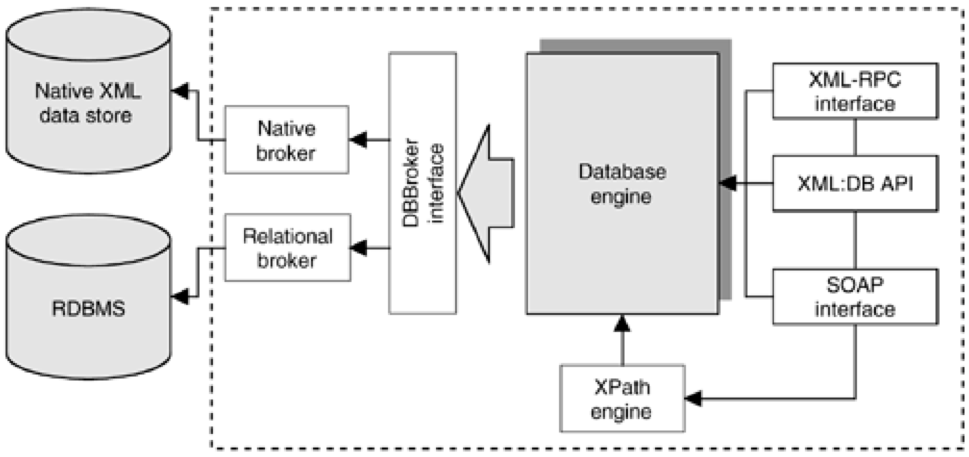
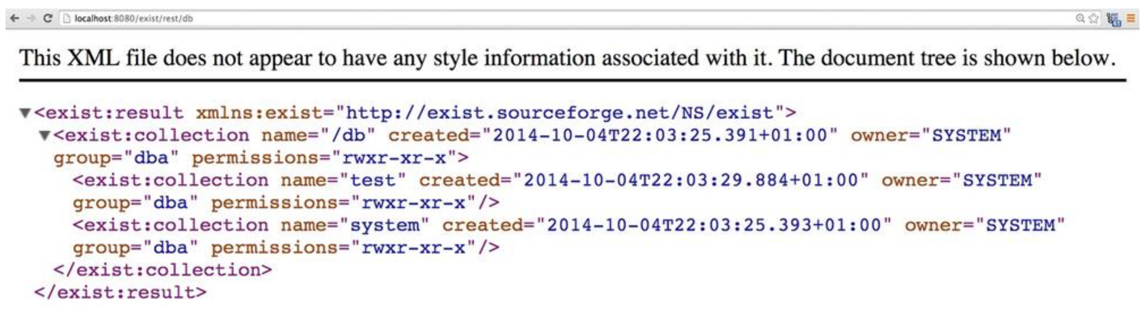


Figure 4. *The eXist architecture [18]*

In the eXist database system, XML documents are not stored as a sequence of XML files on disk due to the inefficiency of this storage format for database operations. Instead, eXist breaks down the data into individual parts and saves it as a collection of binary files that have been optimized for storage space. A highly effective and performant database system is produced as a result of this storage procedure, which also involves the construction of indexes by eXist, and is able to execute complicated queries against the stored XML documents quickly [19].

There are multiple ways to interact with the database: through the interfaces, admin client, or connection via XML:DB driver [20]. Via the Hypertext Transfer Protocol (HTTP), eXist-db offers a REST-style (or, more correctly, RESTful) API that enables rapid and easy access to the database. The API offers ways to alter the database's content and transmits commands to have those commands executed on the database's data [22]. The system will listen for REST requests at http://localhost:8080/exist/rest/ in the default eXIst-db setup.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<exist:result xmlns:exist="http://exist.sourceforge.net/NS/exist">
  ▼<exist:collection name="/db" created="2014-10-04T22:03:25.391+01:00" owner="SYSTEM"
   group="dba" permissions="rwxr-xr-x">
     <exist:collection name="test" created="2014-10-04T22:03:29.884+01:00" owner="SYSTEM"
     group="dba" permissions="rwxr-xr-x"/>
     <exist:collection name="system" created="2014-10-04T22:03:25.393+01:00" owner="SYSTEM"
     group="dba" permissions="rwxr-xr-x"/>
   </exist:collection>
 </exist:result>
```

Figure 5. *Viewing the response of REST request in a web browser [19]*

By making a request with the document's content, the database can save binary and XML documents into collections using the eXist REST Server API. The Internet media type must also be specified in the HTTP Content-Type header. For requests that perform actions in the collection in the database, a username and password are compulsory. For instance, the XML file /tmp/my-doc.xml will be stored in the collection /db/docs/personal when the client for URL (cURL) command is used [19].

```
curl -i -X PUT
  -H 'Content-Type: application/xml'
  --data-binary @/tmp/my-doc.xml
  http://aretter:12345@localhost:8080/exist/rest/db/docs/personal/my-doc.xml
```

Program 4. *A cURL command for storing a document [19]*

Native XML databases are a specialized type of database that store, index, and query XML data directly, without the need to convert it to a relational format. Native XML databases have become increasingly popular in several industries recently, including publishing, healthcare, and government. The native XML databases are also highly scalable and can handle very large volumes of data efficiently. Overall, native XML databases are likely to remain valuable tools for storing and manipulating complex data in the future [23].

2.4.2 Relational database

Relational database stores and organizes data in tables made up of rows and columns. The ability to create relationships between various tables, which enables more effective data organizing and analysis, is one of a relational database's key advantages. For example, a bookstore database might have a table for books, a table for authors. By establishing a relationship between the books and authors tables, it is possible to determine which books belong to which authors [24].

The capacity to manipulate and query the data using Structured Query Language (SQL) is another important factor of a relational database. Relational databases can be created, modified, and their data extracted using the standard programming language SQL. With the support of SQL, a variety of operations can be carried out on the data, including the addition of new records, updates to already existing records, deletions of records, and the retrieval of certain data according to certain criteria [25].

The BLOB (Binary Large OBject) and CLOB (Character Large OBject) data types in relational databases support saving enormous quantities of binary or character data, respectively. Images, movies, audio files, and documents are among the file kinds that these data types are frequently used to store [26]. For instance, a database might have a table for storing employee records, with each record containing a BLOB field for storing the employee's photograph. By storing the photograph in the database, it is possible to easily retrieve and display the photograph along with the rest of the employee's information. However, there are disadvantages of storing files in a database using BLOBs or CLOBs such as size of database backups [26].

2.4.3 XML and relational databases

A file system works well enough for simple applications but struggles to scale when there are hundreds or thousands of documents as a local XML storage system. Problems with concurrency, recovery, security, and usability become unsolvable. The adoption of database management systems (DBMS) relieves some of the limitations associated with file systems, however DBMS options have been limited when it comes to maintaining XML information up until now. Relational DBMSs are typically used in one of two ways: stuffing or shredding [16].

An XML document is enclosed into a single big entity (CLOB or BLOB) column in a relational database. This strategy performs best when the XML documents are saved and retrieved in their whole. Retrieving the contents of XML documents or extracting specific elements, attributes, or subtrees requires scanning each document at runtime, which can be time-consuming [16].

XML data is mapped to and from relational columns and tables during the shredding or deconstructing process. This process divides the XML file into multiple fragments (elements and attributes), which are then inserted into different columns in the database. Due to the complexity of the XML document and normalization procedures, the result of the shredding or deconstructing process can contain hundreds of columns and numerous tables. Accessing these fields and tables becomes overly complicated when the document is being rebuilt since it necessitates complicated queries and multi-table joins. In other cases, such as with digital signatures, it can even be difficult to recreate some documents or maintain the original accuracy of data. With this method, relational data models' formality is likewise injected into XML data formats' adaptability [16].

2.4.4 Hybrid databases

As using CLOB or BLOB data types for storing XML documents is considered inefficient, there are several relational database applications that provide a mechanism for storing XML documents and accommodating complex schemas. Both relational and XML resources can both exist in the same database using the hybrid database. Several applications, such as IBM DB2, Oracle Database, and Microsoft SQL Server, utilize their own engines for storing XML documents in their respective databases. Overall, this system provides an XML data type which enables the definition of columns within a table that store XML values. This pureXML data type in IBM DB2 can store an XML document in a database while maintaining its integrity [16]. The overview of pureXML in IBM DB2 and alternative XML storage options is shown in Figure 4.

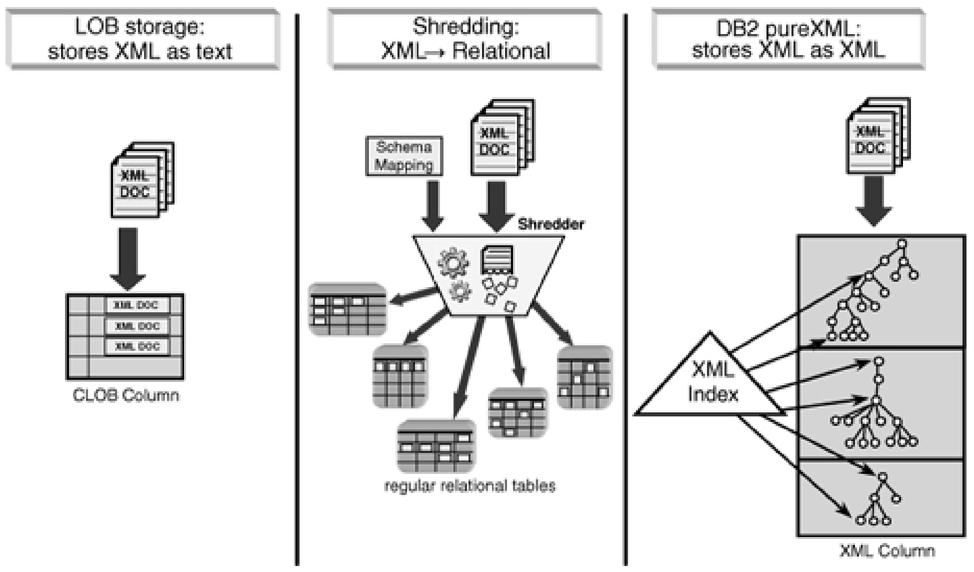


Figure 6. *pureXML and alternative XML storage options [27]*

It should be noted that DB2's pureXML does not transform XML to relational format or store XML data as text in large objects (LOBs). It stores XML documents in a parsed hierarchy when they are loaded or inserted into a column of type XML. It is crucial to remember that each XML document is only ever parsed once when it is appropriate to insert it into an XML column. A major performance advantage of the parsed storage format is that queries and changes can operate on XML data without the need for XML parsing. [27]

2.5 Native XML databases or hybrid ones

A common characteristic of native XML databases is that they do not necessitate the construction of tables or elaborate schemas, as opposed to traditional relational databases [26][28]. Relational databases, in contrast, use two-dimensional tables, which often necessitate the implementation of relationships between tables to accommodate complex data structures. As a result, this can result in a more complicated database design [27].

IBM DB2 and other hybrid databases offer a variety of XML data products and approaches as well as an encompassing vision of the future for XML data management. Users with experience in SQL and conventional relational database programs can use engine-level support that is equivalent to their present toolkit and makes use of their existing expertise when they need to quickly generate XML fragments. Developers will find the function they need along with support for popular APIs, validation calls, and transformation support. They may also create whole XML documents from relational data that already exists or simply load entire XML documents or shred them into relational tables [18].

There are currently several database solutions available that support XML resources. These databases, created by various communities, are available in both open-source and commercial versions. Users can install these programs and load example XML files to use them, preparing the databases for data analysis. The following chapters will go into further detail regarding tools.

3. THE ORIGINAL APPLICATION

The ResourceCatalogueV1 is an innovative tool that facilitates the publication, searching, and viewing of Resource Description (RD) files. Additionally, it provides basic processing capabilities, such as the transformation of the information contained within the RD files into alternative formats, such as human-readable documentation.

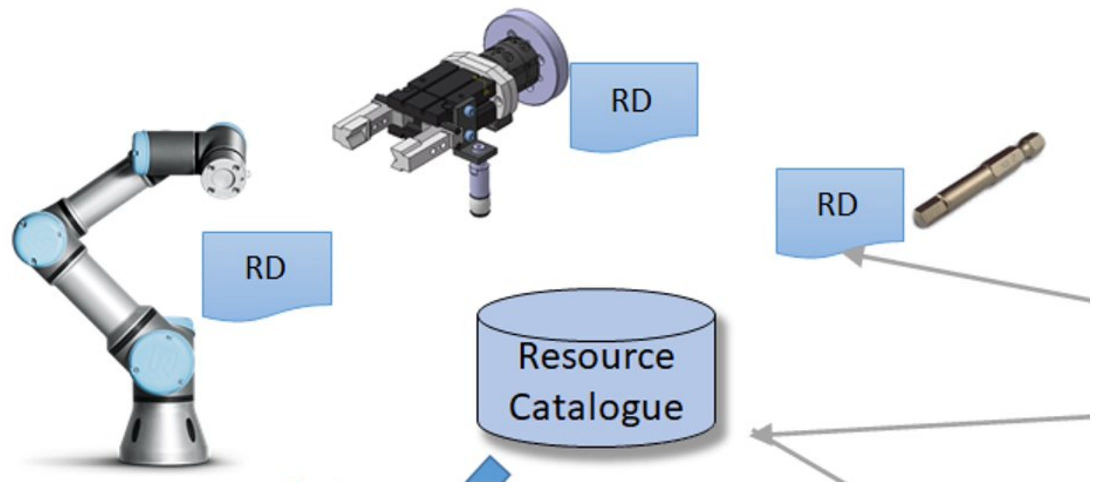


Figure 7. *The platform overview [37]*

The ResourceCatalogueV1 is a server-based application or web-based platform designed for the purpose of publishing and viewing the content of RD files. It serves as a storage, distribution center, and a simple processing tool for RD files, offering a web service interface for integration with other applications. The RD descriptions serve as an information exchange format, defining the properties, capabilities, and interactions of production resources with other resources and systems. Other systems can interact with this service via its RESTful interface or receive information about available resources in the RD format. Multiple instances of the platform can exist, each building its own resource catalogue. Some of these server instances may be for internal use within a company and may be extended for use within the provider network. [36]

3.1 Product functions and user groups

The ResourceCatalogueV1 is envisioned to collaborate with various other tools, such as the Resource Description Editor, which is a standalone software that has the capability to create, modify, and view RD files. The editor will leverage the Resource Catalogue(s) for various purposes, such as generating customized skeleton RD files during the creation of new RD files or for publishing existing RD files. Additionally, many other tools and applications are expected to be closely integrated with or connected to the Resource Catalogue Platform. These tools and applications will utilize the ResourceCatalogueV1 as a source and distribution center for RD files, or as a source of other information, such as the match-making service, which would use the service to generate resource pools. The product functions of the ResourceCatalogueV1 include the following:

- Storage and distribution of RDs with the availability of downloading options.
- Searching suitable RDs based on categorizations, capabilities, and interfaces.
- The implementation of basic RD processing operations, such as the generation of RD skeletons, human-readable documentation, and other modifications.
- Creation of resource pools used in matchmaking.
- Availability of a web service interface (RESTful) that integrates all the features with other applications.

Those functions belong to different groups of users in the ResourceCatalogueV1 as in the Figure 8.

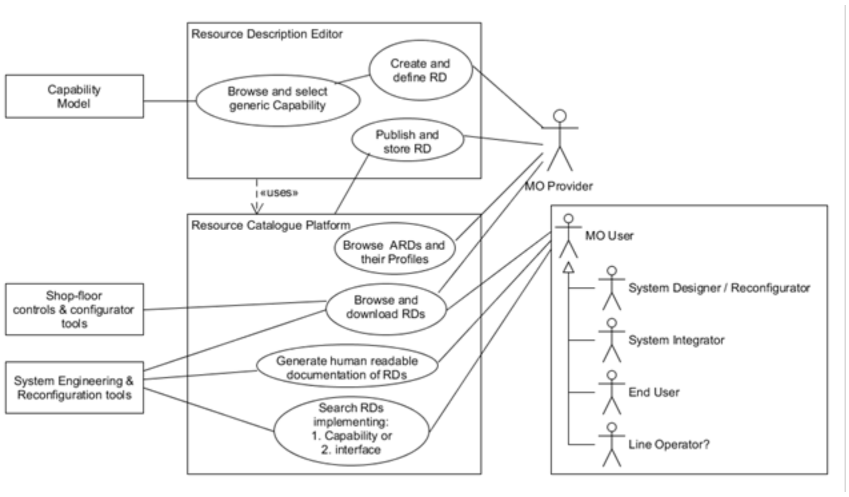

Resource Description Editor
Capability Model
Browse and select generic Capability
Create and define RD
Publish and store RD
«uses»
MO Provider
Resource Catalogue Platform
Browse ARDs and their Profiles
Shop-floor controls & configurator tools
Browse and download RDs
System Engineering & Reconfiguration tools
Generate human readable documentation of RDs
Search RDs implementing: 1. Capability or 2. interface
MO User
System Designer / Reconfigurator
System Integrator
End User
Line Operator?

Figure 8. *Overall Use Case and connected system [36]*

The primary users of the Resource Catalogue Platform are Resource Users, including System Designers, System Integrators, and End Users. These users utilize the platform for searching and selecting appropriate RDs for production purposes. They can access information in the native format or because of data processing through other tools, such as the Matchmaking Web Service and MM GUI application, which are closely integrated with the ResourceCatalogueV1. Factory floor personnel and control engineers can request transformations from the executable capabilities of an RD, depending on the programming environment, language, or communication protocol.

Table 1. *Actors for the ResourceCatalogueV1 [36]*

Actor	**Description**
(Resource) User	Any registered users viewing or utilizing the data stored to the system. They can view and access RDs and ARDs. User could be directly using the web site or access the services with some tool.
Resource Provider	Makes, provides, and publishes the Resource Descriptions for their own resources.
Resource Manager	Manages Resource Descriptions in general. Can add, modify, or remove RDs. Moderates and manages interface standard definitions.
Admin	Administrates the system. Adds/modifies user and company data. Runs background tasks as Generates indexes and back-ups if those needs manual actions. Can modify or remove any resources.

Another user group consists of Resource Providers, who create, modify, and view RDs and use the ResourceCatalogueV1 for publishing and distributing their RDs. They can use the RD Editor for creating and modifying the RD content and can utilize the ResourceCatalogueV1 for creating customized skeleton RD templates.

The Resource Managers are another group of users who distribute RDs through the ResourceCatalogueV1. They create, modify, and view RDs with the assistance of the RDEditor and may need to access and analyze multiple RDs to identify common features and conclude their findings into new or modified RDs. Information retrieval is performed through ResourceCatalogueV1. The Resource Catalogue Platform must identify all users, and each user can have one or multiple roles assigned for accessing different sections or features of the service.

3.2 The application architecture

The approach presented in this application employs a common three-tier architecture in many Java Server Pages (JSP) applications. JSP technology allows for the integration of dynamic content within web pages. The system consists of three components: a Client, a Web Server, and a

Database. The Client, which is typically a web browser or user-facing application, initiates requests [29]. The Web Server, which in this example is Tomcat, utilizes a JSP Engine, which functions as a container for JSP files.

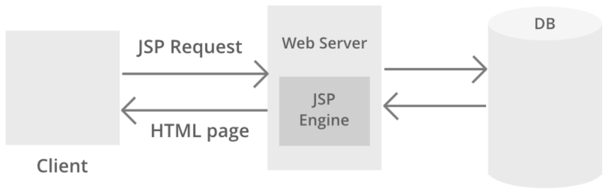


Figure 9. *The architecture of a JSP application [29]*

Apache Tomcat is a free and open-source application server that supports the Jakarta Servlet, Jakarta Expression Language, and WebSocket technologies. It provides the execution of Java code and a fully Java-written HTTP web server environment [30]. Upon receiving a request for a JSP file, the JSP Engine provides the runtime environment necessary for interpreting and executing the file. The web server, by executing the embedded Java code, generates and returns an HTML page to the Client [29].

In the initial design, much emphasis was placed on server-side application features such as database connection, querying performance, and authentication. Due to time constraints and a lack of experience with the required technologies and programming languages, existing technologies were not capable of delivering the required functionalities for this project. To facilitate effective querying features, such as the ability to search for a list of properties in all files based on specific criteria, it is necessary to utilize a database that stores location information. To create the output, the developer must go through the XML files individually and execute multiple XQuery queries.

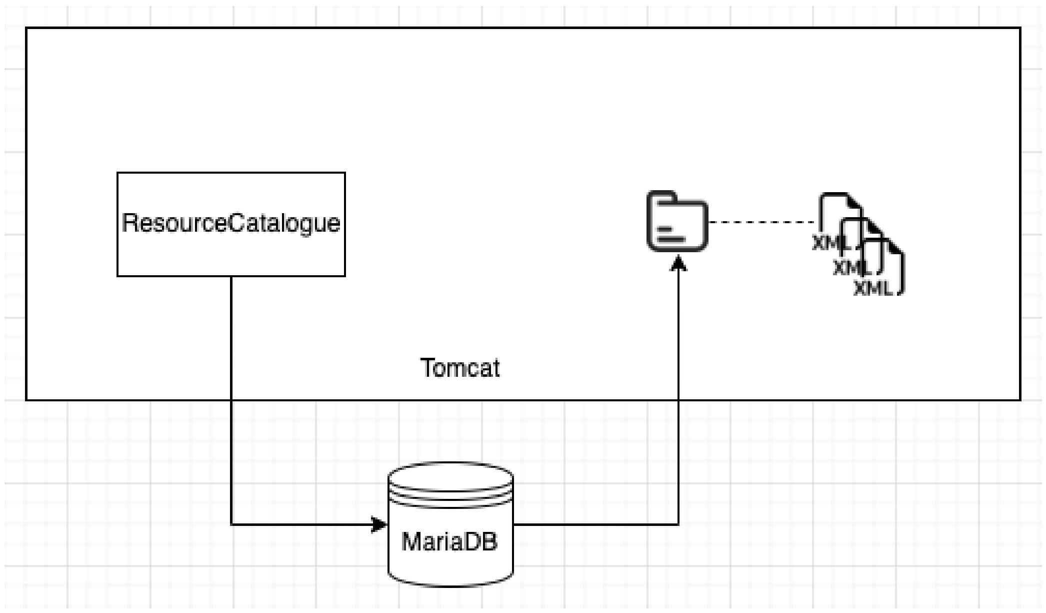


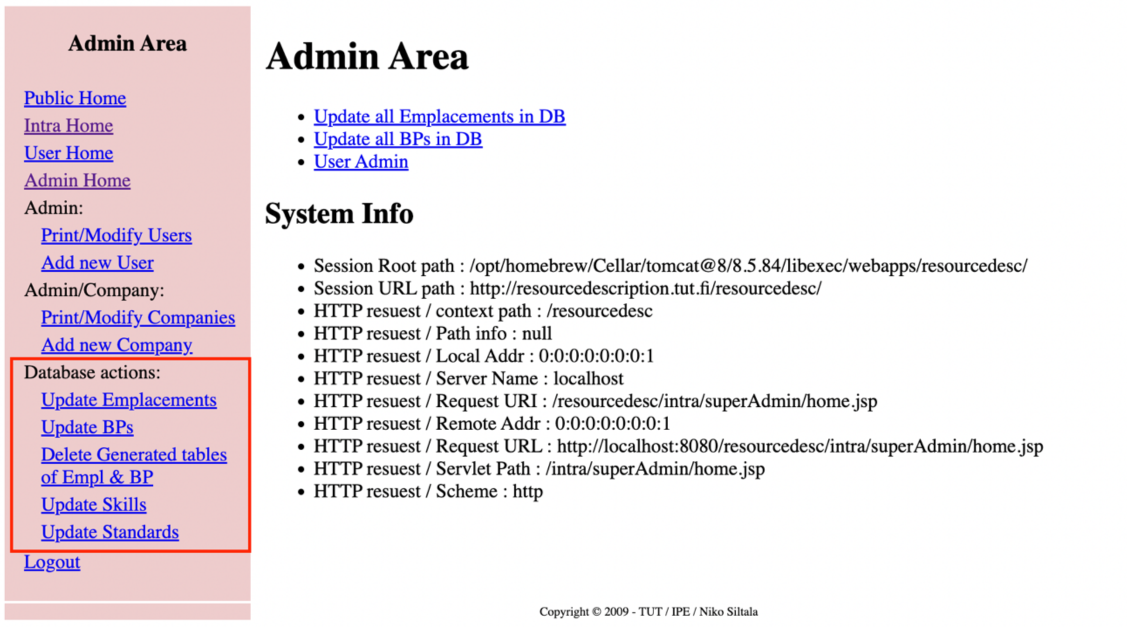
Figure 10. *The ResourceCatalogueV1 simplified architecture*

To provide the information on the relevant XML files within the folder quickly, part of the XML file was retrieved and added to the database table. This is an example of shredding XML files to relational databases mentioned in section 2.4.1. By using this approach, users may access the needed information and filter the results based on their requirements in a fast speed. However, it is possible that the content of the XML file was changed without notifying the developer to run the shredding process, so accuracy cannot be guaranteed.

The results delivered to users could therefore differ from the latest version of the XML file in the folder because of inconsistency. Additionally, if the end-user requested additional information regarding any items in the XML file, changes could be made to the database table definition to provide a comprehensive response to user's query. This introduces a lack of consistency in the system and requires an approach in XML data operation. Since the number of resource files increased over time, the folder option was not practical for long-term applications.

3.3 The application key features

The process of shredding XML content and generating table relationships is the primary strategy for handling resource files in the application. To facilitate this process, the administration team has implemented a scheduled task for handling RD files. Additionally, various actions related to XML files and the database can be performed through the Admin Panel, as shown in Figure 11.



Admin Area

- Update all Emplacements in DB
- Update all BPs in DB
- User Admin

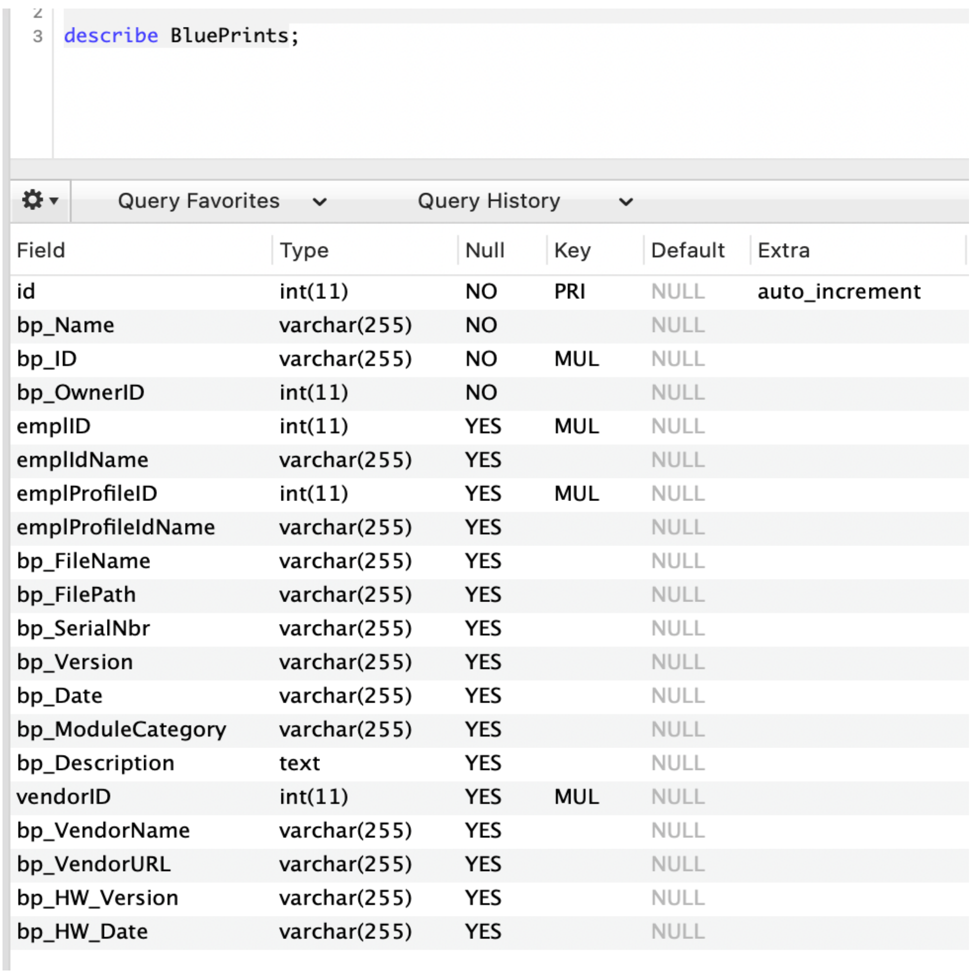
System Info

- Session Root path : /opt/homebrew/Cellar/tomcat@8/8.5.84/libexec/webapps/resourcedesc/
- Session URL path : http://resourcedescription.tut.fi/resourcedesc/
- HTTP resuest / context path : /resourcedesc
- HTTP resuest / Path info : null
- HTTP resuest / Local Addr : 0:0:0:0:0:0:0:1
- HTTP resuest / Server Name : localhost
- HTTP resuest / Request URI : /resourcedesc/intra/superAdmin/home.jsp
- HTTP resuest / Remote Addr : 0:0:0:0:0:0:0:1
- HTTP resuest / Request URL : http://localhost:8080/resourcedesc/intra/superAdmin/home.jsp
- HTTP resuest / Servlet Path : /intra/superAdmin/home.jsp
- HTTP resuest / Scheme : http

Copyright © 2009 - TUT / IPE / Niko Siltala

Figure 11. *Available actions in the admin panel.*

As previously stated, RD files are stored in a specific location on the webserver. In many scenarios, developers need to provide users with the content of RDs, but not all the information is necessary. To address this requirement, a table named Blueprints was designed to keep track of key properties within the RD files. The schema is the centralized data source for all the relevant information required for the application. Based on the resource XML file, the developer selected the necessary general information such as id, name, owner, vendor, serial number, etc. to construct a table definition. This creates a foundation to establish relationships between RDs, Standards and Capabilities following the many-to-many relationship model.



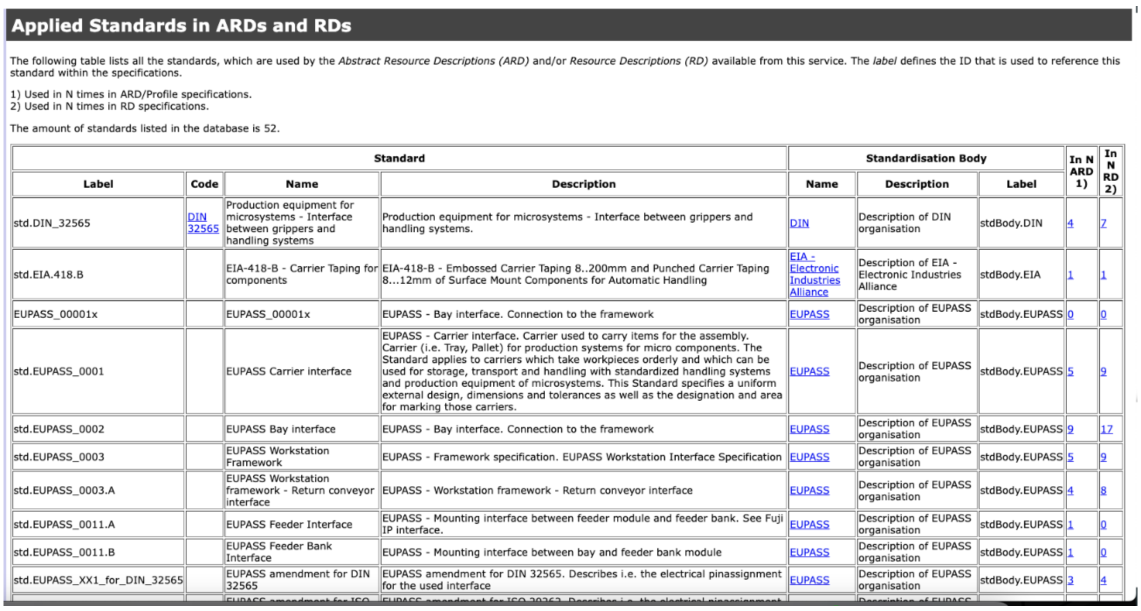
```
describe BluePrints;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
bp_Name	varchar(255)	NO		NULL	
bp_ID	varchar(255)	NO	MUL	NULL	
bp_OwnerID	int(11)	NO		NULL	
emplID	int(11)	YES	MUL	NULL	
emplIdName	varchar(255)	YES		NULL	
emplProfileID	int(11)	YES	MUL	NULL	
emplProfileIdName	varchar(255)	YES		NULL	
bp_FileName	varchar(255)	YES		NULL	
bp_FilePath	varchar(255)	YES		NULL	
bp_SerialNbr	varchar(255)	YES		NULL	
bp_Version	varchar(255)	YES		NULL	
bp_Date	varchar(255)	YES		NULL	
bp_ModuleCategory	varchar(255)	YES		NULL	
bp_Description	text	YES		NULL	
vendorID	int(11)	YES	MUL	NULL	
bp_VendorName	varchar(255)	YES		NULL	
bp_VendorURL	varchar(255)	YES		NULL	
bp_HW_Version	varchar(255)	YES		NULL	
bp_HW_Date	varchar(255)	YES		NULL	

Figure 12. *Important properties of RD files mapped to the table*

The process of extracting information from XML files and updating the database is initiated by the administrator through the admin interface. The function begins by locating the RDs folder in the system and then, for each file, extracting specific key information and adding it to the database or creating new records if they do not exist. Additionally, the function extracts information related to Standards and Standards_Body, updating it internally and creating relationships by adding keys to the Join_tables. This process was implemented to answer the scenario of provision of support for searching suitable RDS based on standard implementation.

As there were previously no data sources for Standards, the developer created a database processing function that, for each RD record in the table, searches for standard IDs and updates the Standards and Standards_Body tables accordingly. The first design focuses on converting the XML file (or the XML-Schema of the XML file) into a collection of SQL statements that create the database's tables to convert the structure of the XML documents to the structure of the database to. Besides, the database also provides information about interface standards for reference in the resource's content. Finally, there is also information on user and company management, who can have access to the internal system through User Interfaces.



Applied Standards in ARDs and RDs

The following table lists all the standards, which are used by the *Abstract Resource Descriptions (ARD)* and/or *Resource Descriptions (RD)* available from this service. The *label* defines the ID that is used to reference this standard within the specifications.

1) Used in N times in ARD/Profile specifications.
2) Used in N times in RD specifications.

The amount of standards listed in the database is 52.

Standard				Standardisation Body			In N ARD 1)	In N RD 2)
Label	**Code**	**Name**	**Description**	**Name**	**Description**	**Label**		
std.DIN_32565	DIN 32565	Production equipment for microsystems - Interface between grippers and handling systems	Production equipment for microsystems - Interface between grippers and handling systems.	DIN	Description of DIN organisation	stdBody.DIN	4	7
std.EIA.418.B		EIA-418-B - Carrier Taping for components	EIA-418-B - Embossed Carrier Taping 8..200mm and Punched Carrier Taping 8...12mm of Surface Mount Components for Automatic Handling	EIA - Electronic Industries Alliance	Description of EIA - Electronic Industries Alliance	stdBody.EIA	1	1
EUPASS_00001x		EUPASS_00001x	EUPASS - Bay interface. Connection to the framework	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	0	0
std.EUPASS_0001		EUPASS Carrier interface	EUPASS - Carrier interface. Carrier used to carry items for the assembly. Carrier (i.e. Tray, Pallet) for production systems for micro components. The Standard applies to carriers which take workpieces orderly and which can be used for storage, transport and handling with standardized handling systems and production equipment of microsystems. This Standard specifies a uniform external design, dimensions and tolerances as well as the designation and area for marking those carriers.	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	5	9
std.EUPASS_0002		EUPASS Bay interface	EUPASS - Bay interface. Connection to the framework	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	9	17
std.EUPASS_0003		EUPASS Workstation Framework	EUPASS - Framework specification. EUPASS Workstation Interface Specification	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	5	9
std.EUPASS_0003.A		EUPASS Workstation framework - Return conveyor interface	EUPASS - Workstation framework - Return conveyor interface	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	4	8
std.EUPASS_0011.A		EUPASS Feeder Interface	EUPASS - Mounting interface between feeder module and feeder bank. See Fuji IP interface.	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	1	0
std.EUPASS_0011.B		EUPASS Feeder Bank Interface	EUPASS - Mounting interface between bay and feeder bank module	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	1	0
std.EUPASS_XX1_for_DIN_32565		EUPASS amendment for DIN 32565	EUPASS amendment for DIN 32565. Describes i.e. the electrical pinassignment for the used interface	EUPASS	Description of EUPASS organisation	stdBody.EUPASS	3	4

Figure 13. *List of standards and its relationship to RDs [2]*

The process of extracting information from XML files and updating the database must be performed by the administrator daily or whenever new XML files are manually added to the centralized folder. This process can be costly in terms of data errors, as well as prone to faults in the function which can lead to corruption of tables and the need for a complete reinstallation.

3.4 The constraints of ResourceCatalogueV1

One key feature that is not yet implemented in the application system is the ability to display a list of uploaded RD files belonging to specific users. From the design illustrated in Figure 8, it is evident that there is a missing connection between the RD table and the user table. However, establishing such a connection is not a straightforward task, due to the following factors:

There is currently no proper way for users to upload RD files on their own. This action can only be performed by individuals who have access to the machine hosting the Tomcat web server. As users can come from external universities or companies, it would not be wise to grant access to the machine to everyone, as it poses security risks and could potentially lead to server crashes due to curious actions.

RD files are external data sources. While it is possible to add new rules for owner definition in the XML files, it would require extensive communication between multiple vendors and data providers to implement correctly. Furthermore, even if owner information is included in the file, the author believes that such relationships should be established through user login to the system.

4. REQUIREMENTS SPECIFICATION

This chapter will provide a detailed discussion of the system requirements for ResourceCatalogueV2. Following the analysis, a specification will be developed.

4.1 ResourceCatalogueV2

ResourceCatalogueV2 is an improved version of TUNI's resource management platform that overcomes the limitation of storing resources in a single folder as discussed in Section 1.1. Additionally, it incorporates previously requested features that were not available in the earlier version. Furthermore, the system facilitates interactions through a list of REST API endpoints, which enables other vendors to integrate with the platform. This subject pertains to the second task outlined in Figure 16, which provides an overview of the new platform.

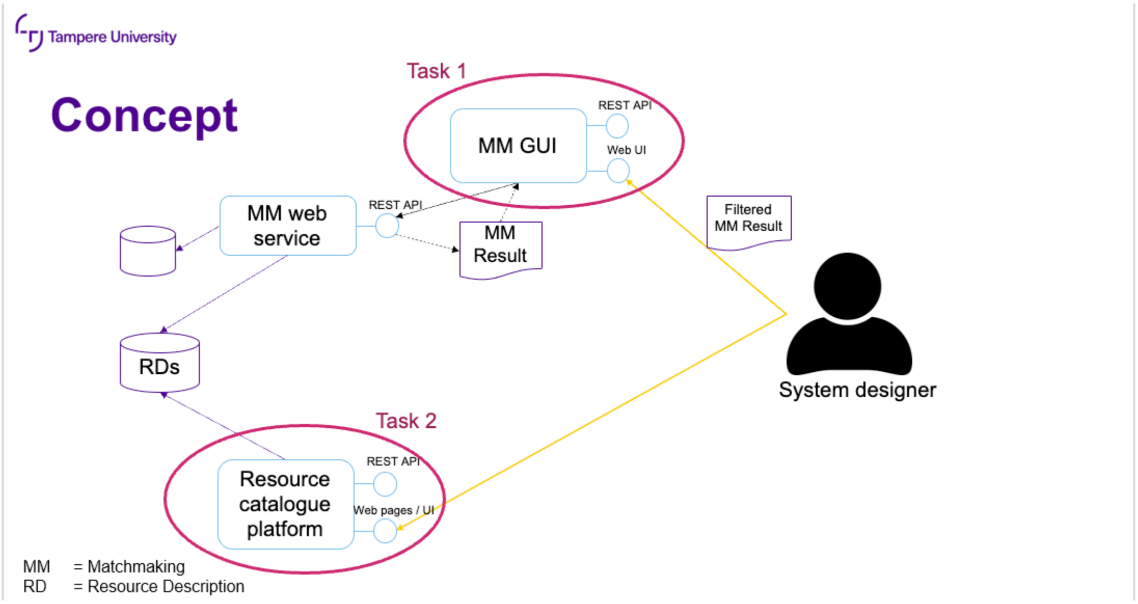


Figure 14. The overview of the new platform [37]

The system is equipped with a hybrid database that permits product owners and users to develop complex queries to obtain all necessary information from resource files. However, the current system does not utilize an XML management system capable of providing advanced XML data querying. Consequently, there is a clear need to enhance the platform by incorporating a database.

4.2 Requirements elicitation

As the product owner, Niko Siltala is responsible for verification that the solution satisfies the ResourceCatalogue's requirements. While Niko has provided documented software specifications for the new solution, relying solely on this information would be an impractical practice. To gather a comprehensive understanding of Niko’s needs and requirements, a thorough study must be conducted. Only then can the solution be developed and tailored to meet his unique needs. Moreover, as Niko is primarily focused on mechanical engineering, his desired features may present several technical challenges. From a practical perspective, obtaining additional requirements through interviews and holding sessions to clarify the requirement specifications will significantly aid in making informed decisions.

To conduct a comprehensive study and identify as many of Niko's needs as possible, the entire requirement elicitation procedure was divided into a subset of stages, which can be summarized as follows:

- Preliminary study of ResourceCatalogue’s source code, data structures of resource files, and conducting interviews with the product owner.
- Identification of end-users for the system being developed and subsequent analysis of their requirements.
- Planning, preparing, and conducting brainstorming sessions.
- Utilizing documented software specifications and designing application API endpoints.
- Elaboration and validation of the identified requirements.

The requirements elicitation phase facilitated the team in obtaining a more comprehensive understanding of Niko's needs. Hence, core topics are addressed in this thesis, which pertain to:

- Applying a database solution in place of a single file folder to offer a more effective means of storing XML files.
- Redesigning the original ResourceCatalogueV1 database to effectively utilize the dedicated data type that has been implemented for XML files.

In the end, we reached a consensus that the prototype for the new application should include the following features:

- Storage and distribution of RDs with the availability of downloading options.

- Provision of support for searching suitable RDs based on categorizations, capabilities, and interfaces.
- Availability of a web service interface (RESTful) that integrates all the features with other applications.

4.3 Scope of the project

The scope of the ResourceCatalogueV2 includes designing and implementing dynamic web pages and web services to enable the storing, retrieving, and processing of RD files. Users should be able to upload RDs, and the system must be equipped with the necessary functionality to search, view, and filter data. Additionally, the project calls for the creation of a REST API to support application integration.

To create an application that provides robust support for database connections, allows for easy development of REST APIs, and can maintain a traditional server-side application, a web framework will be implemented. Technologies such as Java, J2EE, and the Spring Framework are being considered as potential options to accomplish these goals. The frontend development can be realized with a variety of technologies such as React, Angular, AngularJS, AJAX, among others. Regarding the database management system, the project team will decide whether to use MariaDB/MySQL or explore alternative options. As part of the system design, careful consideration will be given to ensure the scalability, reliability, and performance of the selected database system, as it plays a critical role in ensuring the success of the entire ResourceCatalogueV2 solution.

4.4 System requirements

The ResourceCatalogueV2 system requires several software features to store, distribute and manage Resource Descriptions (RDs) effectively. These include the ability to store and distribute RDs and make them available for downloading. The system must also support searching for suitable RDs by categorizations, capabilities, and interfaces, which can be achieved by implementing a search functionality. Additionally, the software must provide simple processing operations for RDs, including generating human-readable documentation, creating RD data structure, and performing other transformations.

Moreover, the ResourceCatalogueV2 system should enable external services like MatchMaking Web service in Figure 15 to access and interact with the resources stored in the database. To accomplish this, the software must be able to identify and group similar resources together, which can be achieved through advanced data structuring and categorization techniques. However, this feature is not critical in the first prototype and considered to be implemented at later phases. Finally, the software must provide a web service interface that is RESTful, which allows for seamless integration with other applications. This feature is essential to allow for easy communication with external systems and to ensure efficient data exchange. Overall, the system must have a comprehensive set of features to effectively store, manage and distribute RDs and to offer web service interfaces for easy integration with other applications.

5. IMPLEMENTATION RESOURCECATA-LOGUEV2

This section will explore the various tools and technologies that are suitable for implementing the ResourceCatalogueV2 project, considering the requirements outlined in the previous section.

5.1 Database solution

In the section discussing the selection of an appropriate database solution for the project, only two types of databases are considered: native XML database and hybrid database.

The use of hybrid databases has become increasingly popular in recent years, particularly in the management of XML files. Unlike traditional file storage methods, hybrid databases offer a more efficient and effective means of storing, retrieving, and manipulating XML data. This is due to their ability to handle structured and unstructured data, as well as their support for powerful querying and indexing capabilities. With the rising importance of XML files in various industries, such as finance, healthcare, and telecommunications, the need for advanced database solutions has become paramount.

Native XML databases have been around since the early 2000s and were originally designed to address the challenges of managing large amounts of XML data. They offer an XML-native data model, which enables the direct storing of XML documents in databases without the need to translate them to relational schemas. These databases are well suited for applications that call for high-performance XML processing since they are designed for effective XML data storage, indexing, and querying.

To determine the most suitable database solution, we evaluate multiple criteria. Firstly, we review the product information, with a focus on the community edition, to understand the available features. Secondly, we examine the documentation, including official product website documentation and relevant books, to ensure that less experienced developers can work comfortably with the chosen database. Additionally, the community surrounding the product is important, as it provides a platform for discussion and problem-solving in case of bugs or errors. Integration with the Spring framework is another crucial factor to consider, as less support would require more effort in managing database connections, leaving fewer hours to work on other high-priority features. Finally, we assess the database's ability to establish relationships between entities, such as the ownership of user files mentioned in Section 3.5, as well as other relationships like user reviews on resource files and supplementary files for a particular resource. We create the table below to outline the differences between the two types of databases.

Table 2. *Comparison of XML databases*

	Native XML database	**Hybrid database**
Documentation	++	+++
Community	Small	Large
Decrease of development effort	No	Yes
Compatibility with a Spring application	No	Yes
Create ownership relations	No	Yes
Maintainability	Person / Group of developers	Company / Organization

Based on the comparison between Native XML Database and Hybrid Database, we recommend the use of Hybrid Database for the ResourceCatalogueV2 project. While Native XML Database provides efficient storage and querying of XML data, the project must also consider the larger design and integration needs. Hybrid Database offers greater flexibility in terms of data model support and integration with other systems, making it a more suitable choice for the project.

In addition to efficient storage of XML resources, the ResourceCatalogueV2 project must also consider the overall design and architecture of the system. Hybrid Database provides the ability to support multiple data models, making it easier to integrate with other systems and services that may not rely solely on XML data. Furthermore, the project requires additional functionality beyond basic data storage and retrieval, such as the ability to perform complex data transformations, and user ownership of XML resources or relationships of supplementary files to the XML files. Hybrid Database can provide greater flexibility in meeting these requirements, while still providing efficient storage and querying of XML resources.

Overall, while Native XML Database may be a good choice for applications that primarily deal with XML data, the ResourceCatalogueV2 project has a broader design in the relationship between XML resources and the application’s users. Its integration needs are better suited to a Hybrid Database. The project can benefit from the flexibility and functionality that a Hybrid Database can provide, while still achieving efficient and effective management of XML resources. At this stage, we have determined that a Hybrid Database is the preferred choice for ResourceCatalogueV2. However, the specific product for implementation has not yet been finalized. Our next step is to identify the most suitable candidate within this category. Currently, the prominent contenders in the Hybrid Database market include Oracle database, IBM DB2, and Microsoft SQL Server.

5.1.1 Oracle database

Oracle Corporation created and promoted Oracle Database, a popular relational database management system (RDBMS). It provides a robust and scalable platform for managing and storing large amounts of structured and semi-structured data. Over the years, Oracle has introduced several features and enhancements to its database technology, including support for XML data. [33]

Oracle XML Database (XML DB) is a built-in component of the Oracle database that is designed to store, manage, and retrieve XML data. As a key component of enterprise database applications, Oracle XML DB offers large-scale XML data storage, simplified data management, and high-performance XML retrieval and processing. Oracle introduced Oracle XML DB as part of the Oracle9iR2 release. It extends the relational storage capabilities of the Oracle database by introducing a new native XML storage type, XMLType. This new storage type serves as the fundamental component of Oracle XML DB for XML data storage, retrieval, and processing. [33]

XMLType makes it easier to manage XML data within the Oracle database by offering a built-in opaque type for handling such data. This type includes predefined member functions that allow for the extraction of XML nodes and fragments. Without native XML storage, XML documents either had to be shredded and stored in multiple relational tables or stored intact as large text/binary objects in CLOBs or BLOBs. The hierarchical nature of XML data often does not align with the tabular format of relational tables, resulting in potential loss of data fidelity. Storing XML as text/binary objects preserves the data but lacks efficient retrieval capabilities for use in high-performance applications. With native XML storage, the processing burden of shredding and re-creating XML from relational tables is eliminated, and built-in performance optimizations for XML-based queries and updates are enabled. [33]

Oracle XML DB differs from pure XML databases as it is seamlessly integrated within the Oracle database, allowing for effective bridging between relational and native XML storage. The use of Oracle XML DB allows organizations to easily leverage the strengths of both relational and native XML storage within a single database. [33]

5.1.2 IBM DB2

When it comes to maintaining and modifying XML data, IBM DB2 is a potent relational database management system with cutting-edge features. With IBM DB2, users can store, query, and update XML data using industry-standard SQL and XQuery languages, as well as take advantage of built-in features such as indexing and validation for optimal performance and data integrity. Additionally, IBM DB2 supports a wide range of XML standards and technologies, including XML Schema, XSLT, and XPath, making it an ideal platform for managing large and complex XML data sets. Whether you are working with structured or semi-structured data, IBM DB2 provides the tools and capabilities developers need to manage XML data with ease and efficiency. [32]

Via its pureXML technology, Db2 offers XML support, enabling client programs to manage XML data inside Db2 tables. The Db2 database system has fully integrated the stored XML data, providing access to it via Db2 features and functionalities. The ability to store XML documents in Db2 tables using the XML data type enables common database operations like creating tables with XML columns, expanding already-existing tables with XML columns, building indexes and triggers on tables with XML columns, and adding, updating, or deleting XML documents. Moreover, the SQL XMLTABLE function can be used to extract data from an XML document and store it in relational tables. [32]

Standard SQL statements can be used to access XML data, or XQuery expressions in SQL with XML extensions can be used to retrieve specific document sections. It is possible to do explicit or implicit XML schema validation to make sure that an XML document's structure, content, and data types are correct. [32]

5.1.3 Microsoft SQL Server

Microsoft SQL Server (MSSQL) is a relational database management system (RDBMS) created by Microsoft Corporation that is frequently used to manage and store enormous amounts of organized and semi-structured data. Designed to meet the needs of organizations of varying sizes, including small businesses as well as large enterprises, this RDBMS offers a robust platform for data management. [34]

It was difficult to store, manage, and manipulate XML files in MSSQL prior to the introduction of SQL Server 2005. In SQL Server 2000, the FOR-XML clause, LOB data type operations, and custom system-stored procedures were all used to implement XML capabilities. However, SQL Server 2005 Edition provides the xml data type elevating the storing and manipulation of XML data to a first-class status in SQL Server. [34]

The xml data type in SQL Server 2008 is recognized for its robust support for XML data storage and manipulation, which offers significant functionality. It can hold untyped XML data as well as typed XML documents and fragments that have been verified against an XML schema collection. This special data type is used for declaring columns in tables, T-SQL variables, parameters, and function returns, among other things. Moreover, it can cast data to and from the xml data type. Each XML instance can contain a maximum of 2.1 GB. [34]

5.1.4 Comparison and choice

Our initial step is to examine the free subscription version for each database, considering that the project has a stringent budget constraint, and that the free version may offer fewer features and capabilities than the commercial edition. Furthermore, we will evaluate whether the free version can support the XML data type, as the application requires building complex SQL queries to retrieve XML elements or attributes and verifying whether an XML element exists in the content.

Additionally, we will assess the suitability of the free license, as the product should be allowed to be used in a production environment. Finally, we will examine the maximum capacity size of the database, as there are approximately 700 XML files to be managed, and if the size of the database is insufficient, it will be necessary to design a database farm to support such a significant number of files. This would be a challenging task to implement and could be difficult for less experienced developers to understand and maintain the database farm. The following figures shows the criteria in regard of three approaches.

Table 3. *Comparison DB2, Oracle DB, SQL Server*

Criteria	**IBM DB2 Community**	**Oracle Express**	**SQL Server Express**
Features	All features	All features	All features
License	Development or production	Development or production	Development or production

Free trial period	Free indefinitely	Free indefinitely	Free indefinitely
Database size	Unlimited	12 GB	10 GB
Processor cores	Up to 4	Up to 2	Up to 4
Allowed memory	16 GB	2 GB	1.4 GB

Finally, compared to the other database, IBM DB2 Community Edition is a free version of DB2, which is a good solution to manage XML files. The system's capacity to use a greater number of cores and consume more instance memory during resource-intensive operations allows it to provide better data processing capabilities over Oracle DB Express and Microsoft SQL Server Express. This allows the system to perform more complex and computationally demanding operations with greater efficiency and speed, making it a suitable choice for applications that require high-performance data processing capabilities. The larger database size supported by IBM DB2 Community Edition means that organizations can store more data without worrying about running out of storage space. In conclusion, if the user wants to do experiments in managing large volumes of XML files, IBM DB2 is the best option compared to the other databases.

5.2 Redesign the original database

One of the goals identified during the requirement elicitation process in Section 4.2 is to redesign the original database to accommodate the specialized data type for XML files in a hybrid database. The application requires an inventive design to simplify the resource management procedure. The initial database architecture, which was inspired from ResourceCatalogueV1, focused on translating the XML file's (or the XML-Schema of the XML file's) structure into a sequence of SQL statements that create the database's tables. Besides, the database also provides information about interface standards for reference in the resource's content. Finally, there is also information on the various user groups mentioned in Section 3, who can have access to the internal system on certain roles.

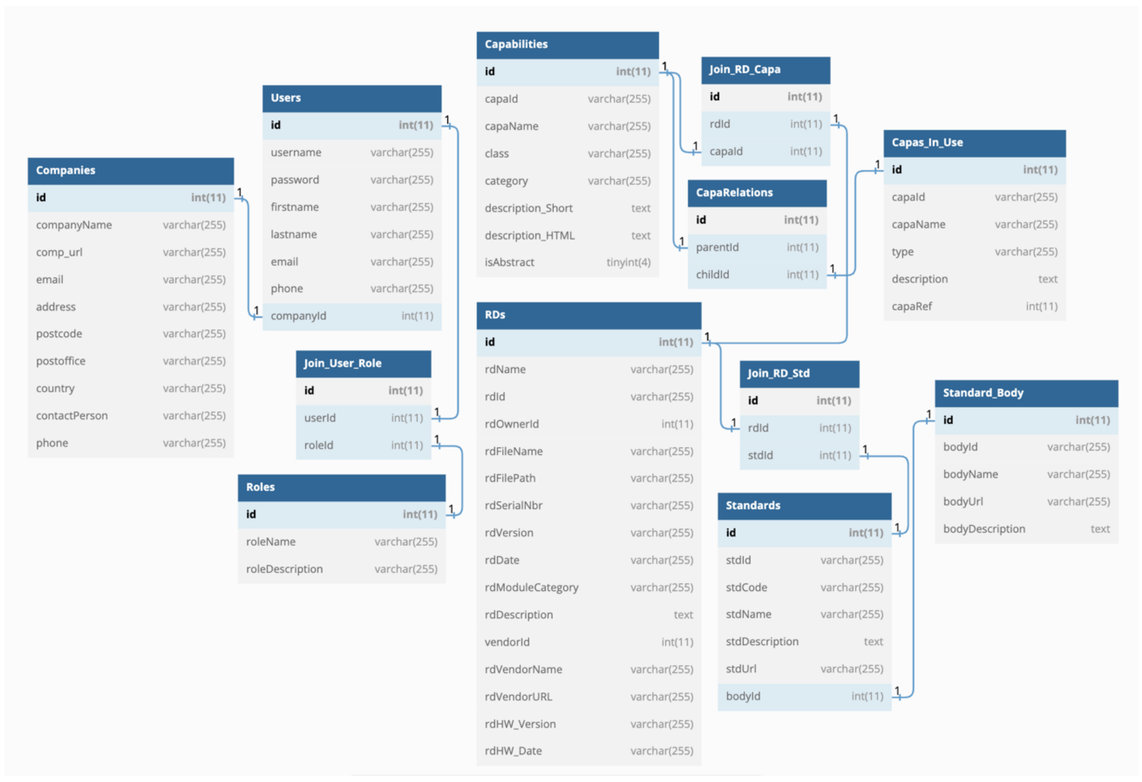


Figure 15. *The design of ResourceCatalogueV1 database.*

The database is composed of two distinct components. The left side contains user information, and the right side contains the relationships between capabilities, standards, and Resource Description (RDs) files. User data, including company and roles in the RD_WS, is provided in the administrative area. However, tables such as those for capabilities, standards, or RDs, as well as Joints tables that display relationships, are generated through the process of shredding XML content and mapping it to tables.

The design is a solution in the scenario that there is a simple data structure of the resource files. As the same database structure is strictly implemented to the current definition of RD file, it cannot be reused for different structure definitions in the same XML format . There can also be the case that the Niko would like to extract other information from the content of resource files. The requirement would force the developer to add new table schemas or potentially redesign the database in the worst scenario to meet the client's needs. Additionally, maintenance is also not an easy task. If the product owner changes the list of interface definitions in the content of a certain resource file, the developer needs to re-process all corresponding information in the database. This is a tedious and time-consuming task. Finally, there is missing relationship information regarding the user and the resource table. It is not easy to know the owner of the resource without adding an additional field that relates the user to the resource.

Due to several limitations of the database, the development team decided to incorporate the IBM DB2's specialized XML data type into the new design, as depicted in the figure below.

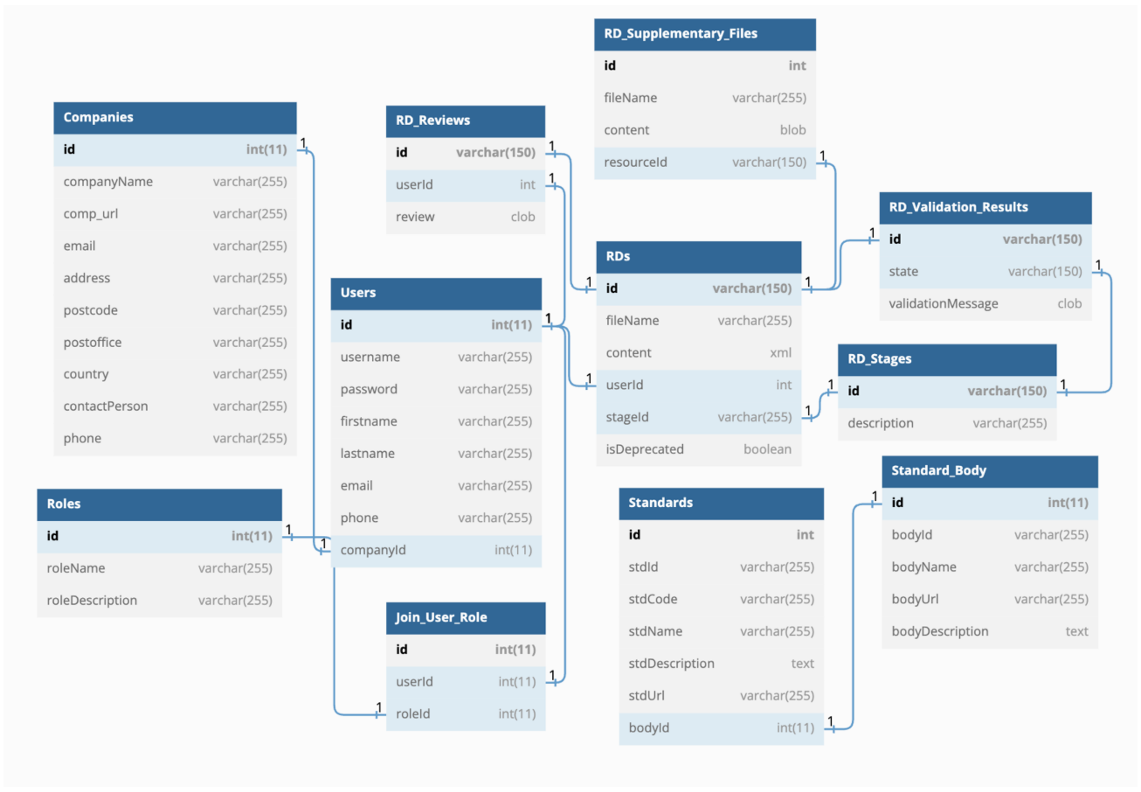


Figure 16. *The database design of the Resource Catalogue V2*

To improve the database design, the development team has implemented the XML datatype feature from IBM DB2, which enables storing XML documents in table columns. This approach simplifies the RDs schema and eliminates the need for complex transformations and foreign key relationships required for storing flat file XML data in the file system. As a result, the Join_RDs_Std table is no longer necessary in the redesigned schema. Moreover, since the Capabilities data comes from an external source, the Capabilities schema is eliminated, and the data can be retrieved from an in-house query library, reducing the effort required to update the Capabilities table periodically. Additionally, the team has introduced a userId column to the RDs schema to establish a one-to-many relationship model, addressing the challenge of missing ownership information for the resource files. The new design for the RDs table differs from the old design by eliminating all columns that were created by extracting specific attributes or elements from the XML files. Instead, the new approach allows for retrieving any data previously stored in those columns directly from the Content column of the RDs table using efficient queries. The focus of the new approach is on establishing connections between resource files and other database tables to address issues like missing ownership information for XML resources.

5.3 ResourceCatalogueV2 architecture

This section describes the architecture and functionalities of the updated version called ResourceCatalogue WebService V2. The system's purpose is to implement missing features and tackle any challenges the original version could not support. The goal of this work is to combine all the features of the prior version while also adding several improvements to increase functionality and usability. The objective of ResourceCatalogueV2's architecture is to fulfill the requirements specified by the requirements in the Section 4. The documentation should give a reader with some experience in the field as well as Niko a good understanding of how the designed system would work.

Prior to delving into the details of the new architecture, this chapter conducts a comprehensive analysis of the original platform architecture, which employs a three-tier architecture model. The presentation tier, the business logic tier, and the data tier are the three divisions that make up this structure, which is frequently used in application development [31].

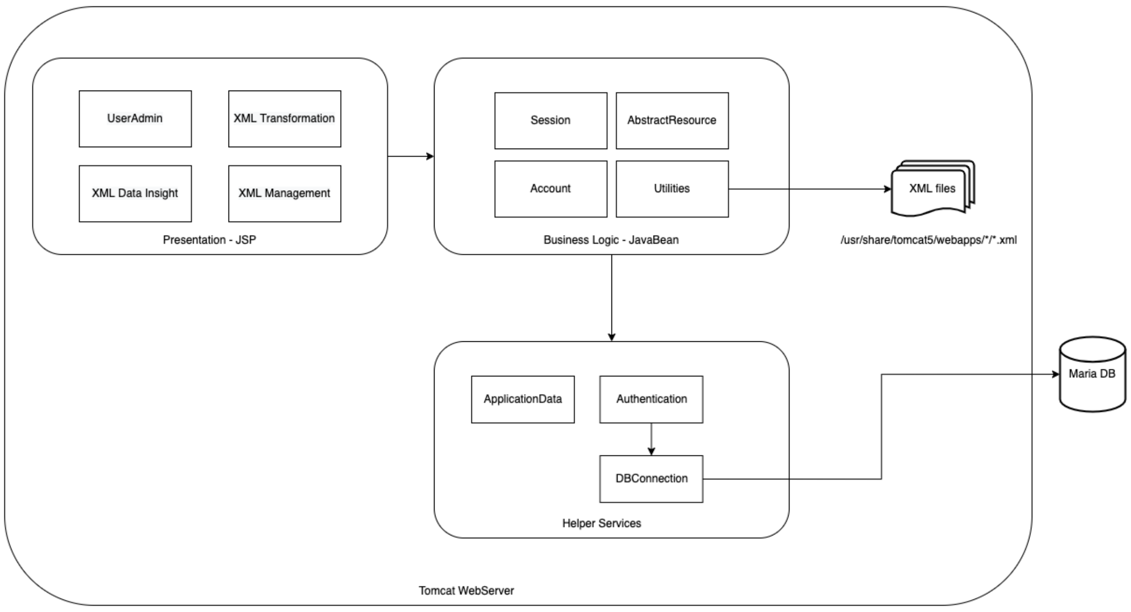


Figure 17. *The structure of ResourceCatalogueV1*

The presentation tier serves as the interface through which the end user interacts with the application, utilizing technologies such as JSP Standard Tag Library (JSTL) and JSP files, as well as stylesheets and JavaScript files, to render the page and deliver information in the form of XML files. The middle tier, also referred to as the logic tier or business logic tier, carries most of the application's functionalities and manages data from both the presentation and data tiers. Additionally, it manipulates the underlying database data as required. Finally, the data tier stores and maintains the information processed by the application and there is no direct communication between this tier and the presentation tier. However, the integration of JSP files, which run

SQL queries within the display layer and directly connect to the database, into the overall application structure may not be optimal.

The design depicted in Figure 17 exemplifies a typical server-side application, where end-users interact with the application's services through the Presentation tier. In addition, the application directly interacts with XML resource files, while certain features communicate with the database. Various services such as managing logged-in sessions, establishing database connections, and authentication have been manually implemented as well. Although the design presented in Figure 17 is simplified, there is a certain level of complexity in the communication between the tiers. For instance, during the discovery phase, the development team noticed that the XML Transformation, which is supposed to be part of the presentation tier, includes implementations that are typically considered part of the business logic tier. The architecture design for the ResourceCatalogueV2, which is noted as Task 2 in Figure 14, has been proposed by the development team and is shown in the figure below.

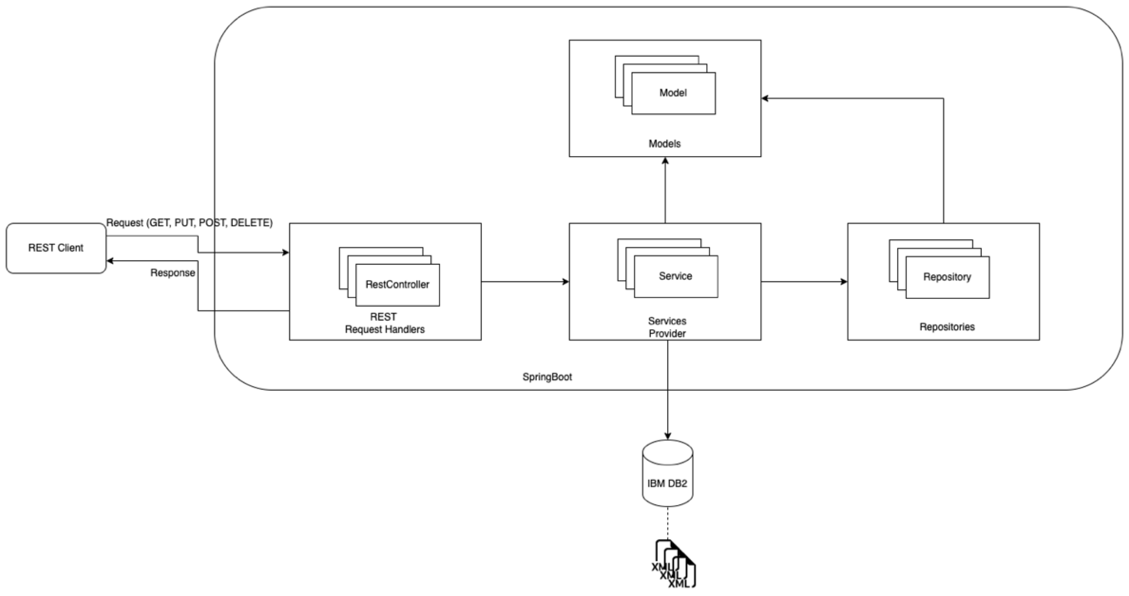


Figure 18. *ResourceCatalogueV2's architecture*

ResourceCatalogueV2 is composed of five main components, including Domain Model, Rest Controller, Service Provider, Repository, and IBM DB2. The Domain Model serves as the object model used within the application and is responsible for transforming data to the front end. The Rest Controller is a SpringBoot application component specialized in making RESTful web services, handling client requests, and returning responses with the support of services defined in the Service Provider. The Service Provider defines the application's functionalities, access methods, and data exchange, independent of any port and its internal workings. The repository, also known as the persistence layer or adapter, abstracts persistence operations such as finding, saving, and deleting records, as well as providing custom actions on database tables. The existence of repository layers allows the development team to concentrate on designing and implementing their service layer to produce the desired value for the response in the controller.

5.4 Accessing IBM DB2 from a SpringBoot application

The ResourceCatalogueV1 required manual implementation for all database interactions, from establishing connections and managing connection pools, to executing SQL queries and processing the results. Spring Boot offers an easy and quick method to set up a project with little configuration and comes with many features and libraries out of the box to minimize those efforts in ResourceCatalogueV2. One important aspect of any web application is the database connection, and in this regard, Spring Boot integrates well with the HikariCP connection pool.

HikariCP is a high-performance Java Database Connectivity (JDBC) connection pool that has become the de facto standard in the Java community. It is lightweight, fast, and configurable, and is well suited for use in modern web applications [35]. When using Spring Boot with HikariCP, configuration is straightforward, and Spring Boot provides several options for tuning the connection pool, such as setting the greatest number of connections, idle timeout, and more.

```
spring.datasource.url=jdbc:db2://localhost:50000/testdb:progressiveStreaming=
2;
spring.datasource.username=db2inst1
spring.datasource.password=password
spring.datasource.driver-class-name=com.ibm.db2.jcc.DB2Driver
spring.datasource.hikari.max-lifetime=4000
spring.datasource.hikari.connection-timeout=4000
spring.datasource.hikari.validation-timeout=4000
spring.datasource.hikari.maximum-pool-size=40
# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
```

Program 5. *Example of setting up connection to database in SpringBoot with HikariCP*

When using the default SQL library from the Java Development Kit to set up a database connection in Java web applications, developers tend to write repetitive methods, resulting in significant effort required to programmatically manage the connection to the database. Utilizing HikariCP is a more beneficial approach as it allows developers to focus on constructing SQL queries and implementing other features. By adopting HikariCP, developers can reduce the amount of code they need to write, leading to improved productivity and reduced development time.

5.5 SQL Queries per use case

In section 4.2, an objective was identified for the conversion of the new web application is provision of support for searching suitable RDs based on categorizations, capabilities, and interfaces. We will examine the objective and assess how different cases were addressed in the new design and implementation.

5.5.1 Extract data from the XML column

The development team's initial aim for the project was to be able to read the content of an XML file that was kept in the XML type column. This is the first assignment because it is the simplest. The first step in the work is to create a query using the database design, where the column for the XML type in the table is called "content." It is crucial that the system be able to access and examine the content of the XML file in this column since it will be used as the basis for other, more difficult tasks that will be implemented later in the project.

```
SELECT content FROM RDs WHERE id = '<resource id>';
```

The command's purpose is to obtain the whole XML file that is kept in the RDs table's content column, using the appropriate ID as a point of reference. This task represents a relatively simple starting point for the project, as it involves querying a specific field in the database and returning its value.

In more complex scenarios, retrieving specific elements, attributes, or lists of elements from XML resources requires more advanced querying techniques. One such technique is using XQuery or XPath to build up simple or queries based on specific cases. This approach is commonly used and supported in IBM DB2, which allows for mixing XQuery or XPath in the SQL query to extract the required data. By using these techniques, it is possible to efficiently retrieve only the relevant parts of the XML content without needing to parse the entire file. This can significantly increase system performance and decrease the amount of processing needed for each query.

To facilitate this process, two templates are provided, one for XPATH and one for XQUERY. These templates can be used to construct queries that retrieve specific elements, attributes, or lists of elements from XML documents stored in the database.

```
SELECT XMLQUERY ('<XPath/XQuery query>' passing content as \"d\") FROM RDs
WHERE id = '<resource id>';
```

These templates provide a simplified approach for constructing XPATH and XQUERY queries to meet specific needs without having to concern oneself with the complexities of the syntax. As an illustration, let us consider the scenario where the team wants to retrieve all implemented standards in a resource file by resource ID. Here is a sample query that mixes XPATH:

```
SELECT XMLQUERY ('$d//*[contains(@id, \"std\")]' passing content as \"d\")
FROM RDs WHERE id = '<resource id>';
```

Another example of a query that can be constructed using the XQUERY template is retrieving a list of values for the attribute named "ifCategory".

```
SELECT XMLQUERY ('
for $d in $CONTENT//*[@ifCategory]
      return $d/@ifCategory/string()\n
') FROM RDs;
```

By using the template constructed for XQUERY/XPATH, developers can easily retrieve specific data from the XML files stored in the database without having to manually construct complex queries.

5.5.2 Verify if elements or attributes exist

In addition to extracting data from XML files, the identification of the presence or absence of elements, attributes, or values within resource files is a crucial component of the search functionality specified by the application requirements. The xmlexists method in IBM DB2 offers an effective means of achieving this functionality. With this function, it is possible to determine whether a specified path expression exists within an XML document and to return a Boolean value indicating its existence. This provides a useful tool for implementing the search and filtering features. The xmlexists method can determine whether a given XQuery expression matches any node in an XML document. To filter rows depending on the presence of a particular XML node, use this function in the WHERE clause of a SQL query. The xmlexists function's fundamental syntax is as follows.

```
XMLEXISTS(XQuery-expression [, document ])
```

In this case, the document is the XML document to which the XQuery expression is applied, and the XQuery-expression is evaluated against the XML document. If the document is not specified, then the function is applied to the XML column in the table. In the ResourceCatalogueV2 project, a scenario is presented where there is a need to locate a resource file that implements a specific identifier.

```
SELECT id, fileName FROM RDs WHERE xmlexists('$d//*:Capability[contains(@id,
\"<id of capability>\")]' passing content as \"d\")
```

The "content" column of the RDs table is an XML element having a Capability tag, and the value of the "id" attribute of the Capability element is the specified capability ID. This SQL query in

IBM DB2 retrieves the id and fileName columns from the RDs table. The XQuery expression is evaluated against the content column of each row in the RDs database using the xmlexists function.

5.5.3 Self-updated query result

The Standards and Standard_body tables in the Figure 15 are entities that needs to be replicated in the new version. The idea of the table is to show a list of Standards and the number of RD files are using each Standard. Based on the old design, the table data is not always up to date because the table's records are the result of migrating XML files to the table. It is not an automatically process whenever the developer adds a file to a folder or a result of scheduled task. To resolve this scenario, we utilize the capability of executing XQUERY/XPATH in the SQL query, we build a simple template in such a way that it will be applied to every single Standards in the database.

```
SELECT s.*, b.*, (SELECT COUNT(id) FROM RDs WHERE
xmlexists('$d//*[contains(@id, $stdId)]' passing content as "d", CAST(s.stdId
as VARCHAR) as "stdId")) as nbrRDs FROM Standards s, Standard_Bodies b WHERE
s.bodyId=b.id ORDER BY b.bodyId, s.stdId;
```

The query retrieves data from two tables, Standards and RDs. The goal of this query is to retrieve all columns from the Standards table along with the number of resources (nbrRDs) aligned to each standard, sorted by the bodyId and stdId columns. The query begins with a SELECT statement that specifies the columns to be retrieved from the Standards table. The asterisk (*) indicates that all columns should be selected. Next, a subquery is used to count the number of records in the RDs table that contain a specific standard ID. This subquery is enclosed in parentheses and preceded by the alias nbrRDs. In the XML content column of the RDs table, the standard ID is looked for using the xmlexists function. The function accepts two arguments: a passing clause that defines the values of any variables used in the search pattern and a text expression that specifies the search pattern. In this case, the search pattern is '$d//*[contains(@id, $stdId)]' and the passing clause includes two variables: 'content as "d"' and 'CAST(std.stdId as VARCHAR) as "stdId"'. The content variable represents the XML content of the RDs table, and the stdId variable represents the standard ID from the Standards table.

In summary, the query retrieves data from the Standards table along with the number of resources aligned to each standard, using a subquery to count the number of records in the RDs table that contain a specific standard ID.

6. EVALUATION

This section provides evaluations of the study, along with a comprehensive discussion of the prototype that was developed.

6.1 Evaluation of the study

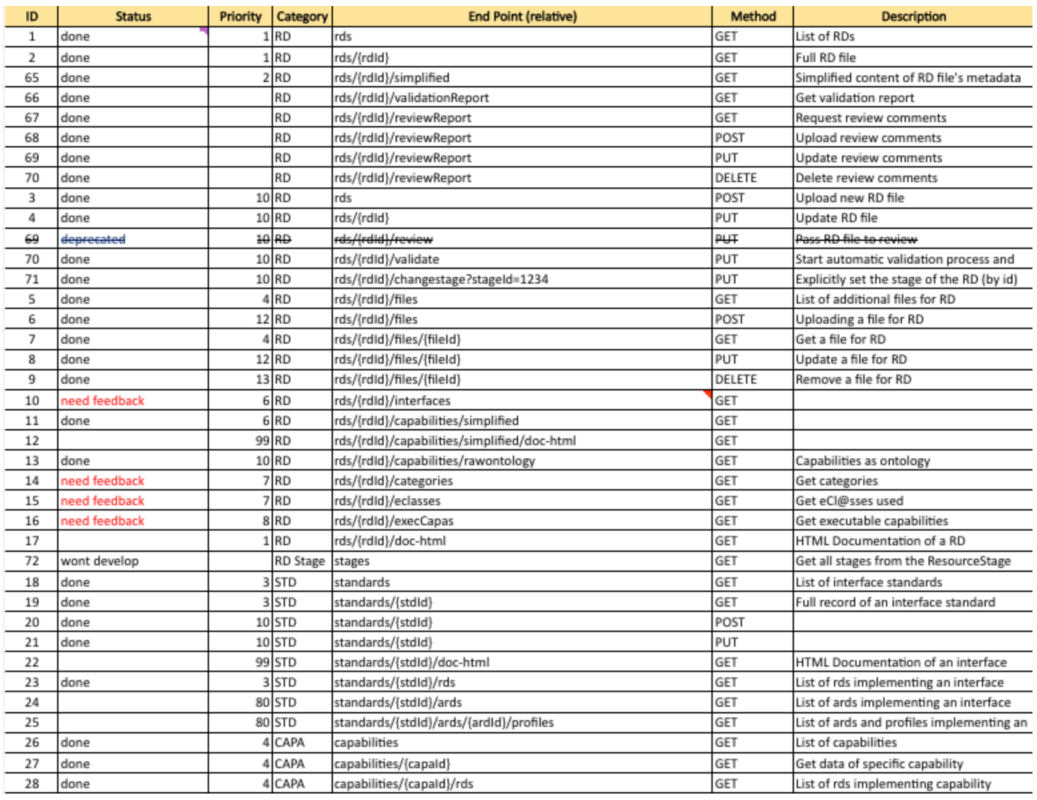
The goal of this study was to examine which hybrid databases would be best for managing XML resources. The aim was to identify the best approaches, technologies, and tools for designing and implementing a prototype that meets the specified requirements. To achieve this, the study delved into the various concepts related to hybrid database management, particularly in the context of XML resources.

The requirements specification phase proved to be the most challenging aspect of this work, as it was crucial to the successful implementation of a hybrid database for XML resource management. To elicit the necessary requirements, a hybrid approach was adopted, which involved analyzing software specifications, investigating the original source code, and conducting interview sessions with the product owner. This process, nonetheless, was not without its challenges, such as the product owner's insufficient understanding of software development and the challenges to the author of this thesis in the understanding of the concept of ResourceCatalogue platform in Figure 7.

The study intended to identify the most suitable hybrid databases for managing XML resources, considering factors such as performance, scalability, and flexibility. Ultimately, the study aimed to equip its readers with the necessary knowledge and skills to design and implement a functional prototype for managing XML resources using hybrid databases.

6.2 Evaluation of the prototype

The implementation of the prototype confirmed the feasibility of creating an XML resource management solution as a service for Open-Digital-Industrial and Networking project (ODIN). To evaluate the functionalities of ResourceCatalogueV2, a series of sessions were scheduled with the product owner to review the APIs outlined in the application specifications. The developed prototype underwent testing and met all the diverse requirements identified during the requirements gathering phase. Furthermore, it was fully functional with the hardware and software configuration selected for the study. These meetings allowed us to document the status of the application APIs, which is presented in the following figure.



ID	Status	Priority	Category	End Point (relative)	Method	Description
1	done	1	RD	rds	GET	List of RDs
2	done	1	RD	rds/{rdId}	GET	Full RD file
65	done	2	RD	rds/{rdId}/simplified	GET	Simplified content of RD file's metadata
66	done		RD	rds/{rdId}/validationReport	GET	Get validation report
67	done		RD	rds/{rdId}/reviewReport	GET	Request review comments
68	done		RD	rds/{rdId}/reviewReport	POST	Upload review comments
69	done		RD	rds/{rdId}/reviewReport	PUT	Update review comments
70	done		RD	rds/{rdId}/reviewReport	DELETE	Delete review comments
3	done	10	RD	rds	POST	Upload new RD file
4	done	10	RD	rds/{rdId}	PUT	Update RD file
~~69~~	~~deprecated~~	~~10~~	~~RD~~	~~rds/{rdId}/review~~	~~PUT~~	~~Pass RD file to review~~
70	done	10	RD	rds/{rdId}/validate	PUT	Start automatic validation process and
71	done	10	RD	rds/{rdId}/changestage?stageId=1234	PUT	Explicitly set the stage of the RD (by id)
5	done	4	RD	rds/{rdId}/files	GET	List of additional files for RD
6	done	12	RD	rds/{rdId}/files	POST	Uploading a file for RD
7	done	4	RD	rds/{rdId}/files/{fileId}	GET	Get a file for RD
8	done	12	RD	rds/{rdId}/files/{fileId}	PUT	Update a file for RD
9	done	13	RD	rds/{rdId}/files/{fileId}	DELETE	Remove a file for RD
10	need feedback	6	RD	rds/{rdId}/interfaces	GET	
11	done	6	RD	rds/{rdId}/capabilities/simplified	GET	
12		99	RD	rds/{rdId}/capabilities/simplified/doc-html	GET	
13	done	10	RD	rds/{rdId}/capabilities/rawontology	GET	Capabilities as ontology
14	need feedback	7	RD	rds/{rdId}/categories	GET	Get categories
15	need feedback	7	RD	rds/{rdId}/eclasses	GET	Get eCl@sses used
16	need feedback	8	RD	rds/{rdId}/execCapas	GET	Get executable capabilities
17		1	RD	rds/{rdId}/doc-html	GET	HTML Documentation of a RD
72	wont develop		RD Stage	stages	GET	Get all stages from the ResourceStage
18	done	3	STD	standards	GET	List of interface standards
19	done	3	STD	standards/{stdId}	GET	Full record of an interface standard
20	done	10	STD	standards/{stdId}	POST	
21	done	10	STD	standards/{stdId}	PUT	
22		99	STD	standards/{stdId}/doc-html	GET	HTML Documentation of an interface
23	done	3	STD	standards/{stdId}/rds	GET	List of rds implementing an interface
24		80	STD	standards/{stdId}/ards	GET	List of ards implementing an interface
25		80	STD	standards/{stdId}/ards/{ardId}/profiles	GET	List of ards and profiles implementing an
26	done	4	CAPA	capabilities	GET	List of capabilities
27	done	4	CAPA	capabilities/{capaId}	GET	Get data of specific capability
28	done	4	CAPA	capabilities/{capaId}/rds	GET	List of rds implementing capability

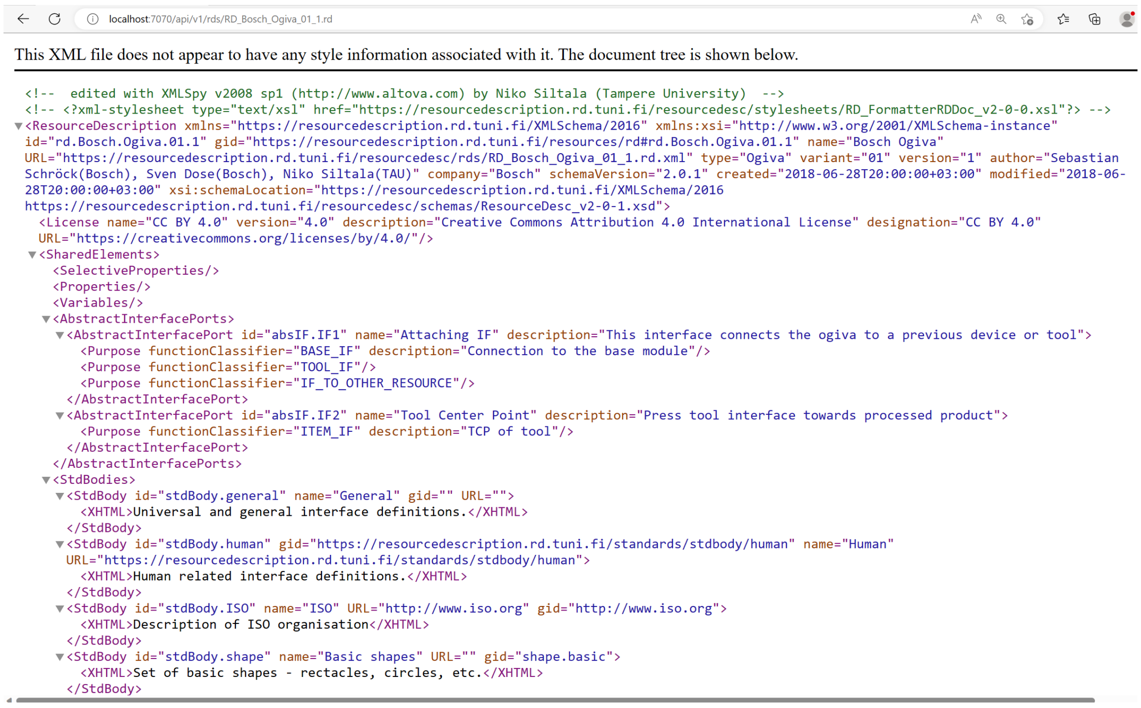
Figure 19. *Summary of ResourceCatalogueV2 APIs*

Upon the completion of the project, a total of 72 API endpoints were identified. Out of these, 47 endpoints were successfully implemented, while 4 endpoints required additional feedback and 2 necessitated further clarification. One endpoint was deprecated, and another was designated as not to be developed. A group of 15 endpoints was excluded from the review process, as they fell beyond the scope of the development objectives.

In the following section, each of the goals established for the conversion of the pre-existing web application will be scrutinized to determine the extent to which they were addressed in the new design and implementation.

6.2.1 Goal#1 – Accessing the resources

The Resource User is empowered to access specific resources (such as XML Schemas, RDs, XSLTs) directly if the URL and filename are known. The ResourceCatalogueV2 Platform serves as the host for these resources. This functionality facilitates seamless application integration, allowing links from a known RD to function efficiently and enabling access to resources for XML validation purposes. Notably, these resources are freely accessible and do not require authentication.

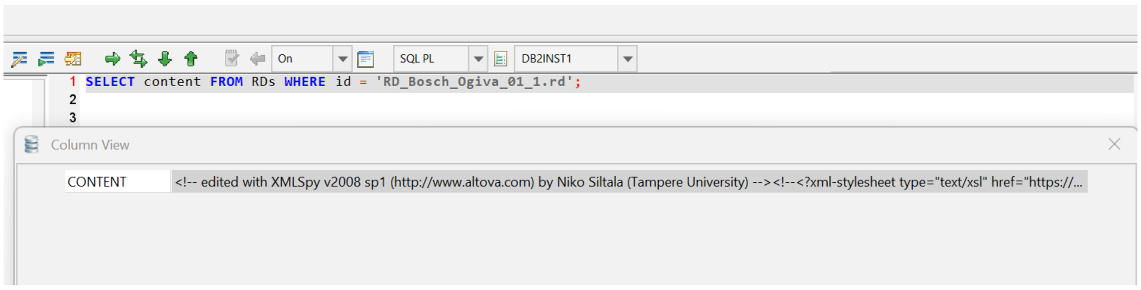

localhost:7070/api/v1/rds/RD_Bosch_Ogiva_01_1.rd

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--  edited with XMLSpy v2008 sp1 (http://www.altova.com) by Niko Siltala (Tampere University)  -->
<!-- <?xml-stylesheet type="text/xsl" href="https://resourcedescription.rd.tuni.fi/resourcedesc/stylesheets/RD_FormatterRDDoc_v2-0-0.xsl"?> -->
▼<ResourceDescription xmlns="https://resourcedescription.rd.tuni.fi/XMLSchema/2016" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 id="rd.Bosch.Ogiva.01.1" gid="https://resourcedescription.rd.tuni.fi/resources/rd#rd.Bosch.Ogiva.01.1" name="Bosch Ogiva"
 URL="https://resourcedescription.rd.tuni.fi/resourcedesc/rds/RD_Bosch_Ogiva_01_1.rd.xml" type="Ogiva" variant="01" version="1" author="Sebastian
 Schröck(Bosch), Sven Dose(Bosch), Niko Siltala(TAU)" company="Bosch" schemaVersion="2.0.1" created="2018-06-28T20:00:00+03:00" modified="2018-06-
 28T20:00:00+03:00" xsi:schemaLocation="https://resourcedescription.rd.tuni.fi/XMLSchema/2016
 https://resourcedescription.rd.tuni.fi/resourcedesc/schemas/ResourceDesc_v2-0-1.xsd">
   <License name="CC BY 4.0" version="4.0" description="Creative Commons Attribution 4.0 International License" designation="CC BY 4.0"
   URL="https://creativecommons.org/licenses/by/4.0/"/>
 ▼<SharedElements>
     <SelectiveProperties/>
     <Properties/>
     <Variables/>
   ▼<AbstractInterfacePorts>
     ▼<AbstractInterfacePort id="absIF.IF1" name="Attaching IF" description="This interface connects the ogiva to a previous device or tool">
         <Purpose functionClassifier="BASE_IF" description="Connection to the base module"/>
         <Purpose functionClassifier="TOOL_IF"/>
         <Purpose functionClassifier="IF_TO_OTHER_RESOURCE"/>
       </AbstractInterfacePort>
     ▼<AbstractInterfacePort id="absIF.IF2" name="Tool Center Point" description="Press tool interface towards processed product">
         <Purpose functionClassifier="ITEM_IF" description="TCP of tool"/>
       </AbstractInterfacePort>
     </AbstractInterfacePorts>
   ▼<StdBodies>
     ▼<StdBody id="stdBody.general" name="General" gid="" URL="">
         <XHTML>Universal and general interface definitions.</XHTML>
       </StdBody>
     ▼<StdBody id="stdBody.human" gid="https://resourcedescription.rd.tuni.fi/standards/stdbody/human" name="Human"
       URL="https://resourcedescription.rd.tuni.fi/standards/stdbody/human">
         <XHTML>Human related interface definitions.</XHTML>
       </StdBody>
     ▼<StdBody id="stdBody.ISO" name="ISO" URL="http://www.iso.org" gid="http://www.iso.org">
         <XHTML>Description of ISO organisation</XHTML>
       </StdBody>
     ▼<StdBody id="stdBody.shape" name="Basic shapes" URL="" gid="shape.basic">
         <XHTML>Set of basic shapes - rectacles, circles, etc.</XHTML>
       </StdBody>
```

Figure 20. *The entire contents of an XML resource can be obtained through an API call.*

Based on the information provided in Figure 20, the URL for accessing the full content of an XML file by its ID is /api/v1/rds/RD_Bosch_Ogiva_01_1.rd, where the ID of the resource file is "RD_Bosch_Ogiva_01_1.rd". Using this URL, we can create an SQL command from the template for extracting the entire XML document based on an ID, as mentioned in section 5.1.1. The result of this SQL command can be observed in the following figure.



```
SELECT content FROM RDs WHERE id = 'RD_Bosch_Ogiva_01_1.rd';
```

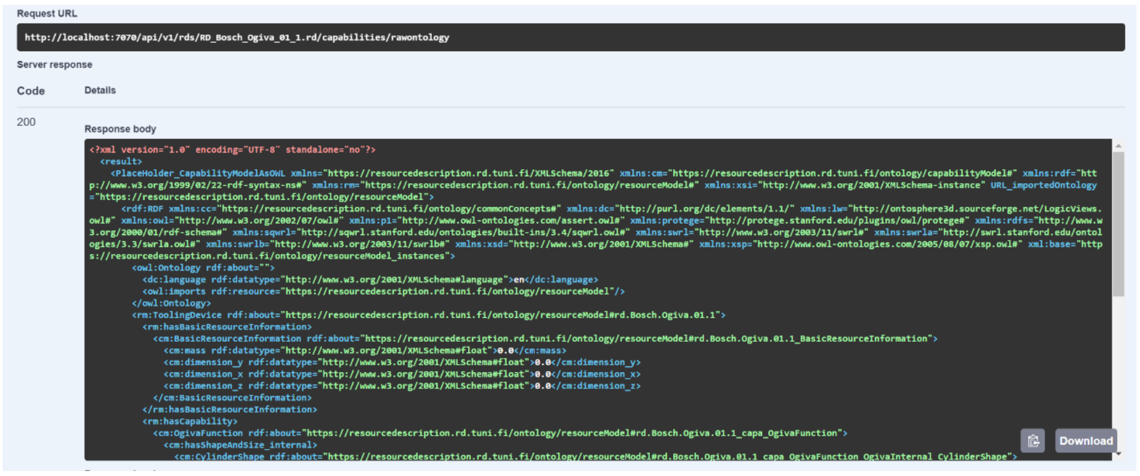
CONTENT <!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Niko Siltala (Tampere University) --><!--<?xml-stylesheet type="text/xsl" href="https://...

Figure 21. *A result of getting XML document by ID*

Viewing the Content table can be varying for different database clients. We also observe that the XML file was minified in the view after the insertion step. However, there are no losses in the content when compared to the original one, so we decided not to investigate further.

6.2.2 Goal#2 – Querying the resources

The ResourceCatalogueV2 offers several search functionalities related to Resource Descriptions (RDs). These searches include identifying RDs that implement a specific capability, RDs that implement a specific interface standard, and the vendor(s) providing the resources. Additionally, a search can be performed to identify a resource that includes a specific element or attribute. The search operation functions as an "AND" operation, which means that one or more capabilities, interface standards, vendor(s), or resource category(ies) can be used as search criteria. The search results will only include resources that satisfy all specified conditions. Finally, the search results can be sorted by the name of the RD, capability name, interface standard ID or name, standard organization, vendor name, or resource category.


Request URL

```
http://localhost:7070/api/v1/rds/RD_Bosch_Ogiva_01_1.rd/capabilities/rawontology
```

Server response

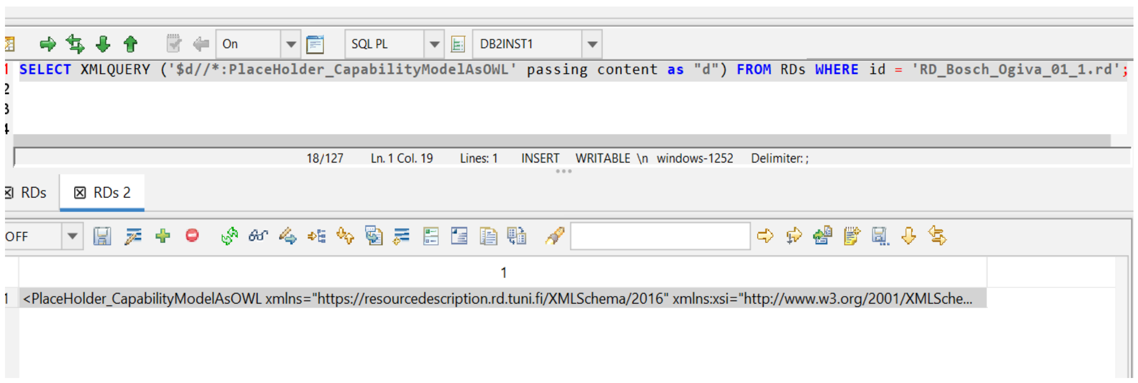
Code Details

200 Response body

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <result>
    <PlaceHolder_CapabilityModelAsOWL xmlns="https://resourcedescription.rd.tuni.fi/XMLSchema/2016" xmlns:cm="https://resourcedescription.rd.tuni.fi/ontology/capabilityModel#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rm="https://resourcedescription.rd.tuni.fi/ontology/resourceModel#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" URL_importedOntology="https://resourcedescription.rd.tuni.fi/ontology/resourceModel">
      <rdf:RDF xmlns:cc="https://resourcedescription.rd.tuni.fi/ontology/commonConcepts#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:lw="http://ontosphere3d.sourceforge.net/LogicViews.owl#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:p1="http://www.owl-ontologies.com/assert.owl#" xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#" xmlns:swrl="http://www.w3.org/2003/11/swrl#" xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#" xmlns:swrlb="http://www.w3.org/2003/11/swrlb#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#" xml:base="https://resourcedescription.rd.tuni.fi/ontology/resourceModel_instances">
        <owl:Ontology rdf:about="">
          <dc:language rdf:datatype="http://www.w3.org/2001/XMLSchema#language">en</dc:language>
          <owl:imports rdf:resource="https://resourcedescription.rd.tuni.fi/ontology/resourceModel"/>
        </owl:Ontology>
        <rm:ToolingDevice rdf:about="https://resourcedescription.rd.tuni.fi/ontology/resourceModel#rd.Bosch.Ogiva.01.1">
          <rm:hasBasicResourceInformation>
            <cm:BasicResourceInformation rdf:about="https://resourcedescription.rd.tuni.fi/ontology/resourceModel#rd.Bosch.Ogiva.01.1_BasicResourceInformation">
              <cm:mass rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.0</cm:mass>
              <cm:dimension_y rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.0</cm:dimension_y>
              <cm:dimension_x rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.0</cm:dimension_x>
              <cm:dimension_z rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.0</cm:dimension_z>
            </cm:BasicResourceInformation>
          </rm:hasBasicResourceInformation>
          <rm:hasCapability>
            <cm:OgivaFunction rdf:about="https://resourcedescription.rd.tuni.fi/ontology/resourceModel#rd.Bosch.Ogiva.01.1_capa_OgivaFunction">
              <cm:hasShapeAndSize_internal>
                <cm:CylinderShape rdf:about="https://resourcedescription.rd.tuni.fi/ontology/resourceModel#rd.Bosch.Ogiva.01.1_capa_OgivaFunction_OgivaInternal_CylinderShape">
```

Figure 22. *An illustration of the output obtained by querying an XML attribute through an API call.*

The output displayed in Figure 22 is the result of the API call, which was obtained using the SQL command mentioned in section 5.1.1. However, the focus of this scenario was primarily on the contents of the "Content" column. Following the approach outlined in section 6.2.1, we created an SQL command that searches for XML elements within the XML document based on their ID.



```
SELECT XMLQUERY ('$d//*:PlaceHolder_CapabilityModelAsOWL' passing content as "d") FROM RDs WHERE id = 'RD_Bosch_Ogiva_01_1.rd';
```

18/127 Ln. 1 Col. 19 Lines: 1 INSERT WRITABLE \n windows-1252 Delimiter: ;
RDs
RDs 2
OFF

```
   1
1  <PlaceHolder_CapabilityModelAsOWL xmlns="https://resourcedescription.rd.tuni.fi/XMLSchema/2016" xmlns:xsi="http://www.w3.org/2001/XMLSche...
```

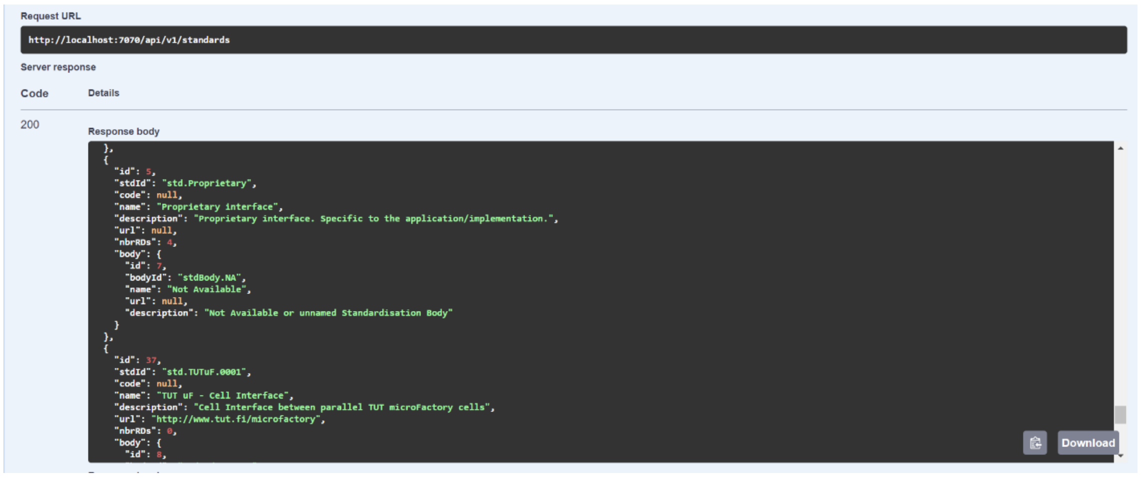
Figure 23. *A result of mixing XPath & SQL*

In the initial prototype, various APIs were developed to retrieve commonly used data from the XML resources. While the requirement was to have a general search API that could combine different query APIs, we encountered a challenge due to the structure of the response data from the query API being a single source. To address this issue, we delegated the task of identifying common elements in multiple arrays and displaying the results to the frontend team.

6.2.3 Goal#3 – Browse Interface Standard Specifications

The tables "Standard" and "Standard_Body" were migrated from the original database to IBM DB2 without any modifications. These tables store standardized interface documentations from various organizations, such as the International Organization for Standardization (ISO), Evolvable Ultra-Precision Assembly Systems (EUPASS), and TUT µFactory [5]. In this study, each RD file implements a list of Interface Standards in the form of XML elements. In the previous database design, the implementation status of RD's Interface Standards was mapped to relational tables to quickly provide the result. However, that approach was not concise and did not provide real-time updates when new RD resources were added to the database. Niko identified this as a challenge in providing an up-to-date count of RD resources implementing Interface Standards in a simple manner.

We developed an API for Niko that retrieves all rows from the Standard and Standard_Body tables. The API also includes the latest count of RD files that implement the corresponding Interface Standard, as shown in the accompanying screenshot.



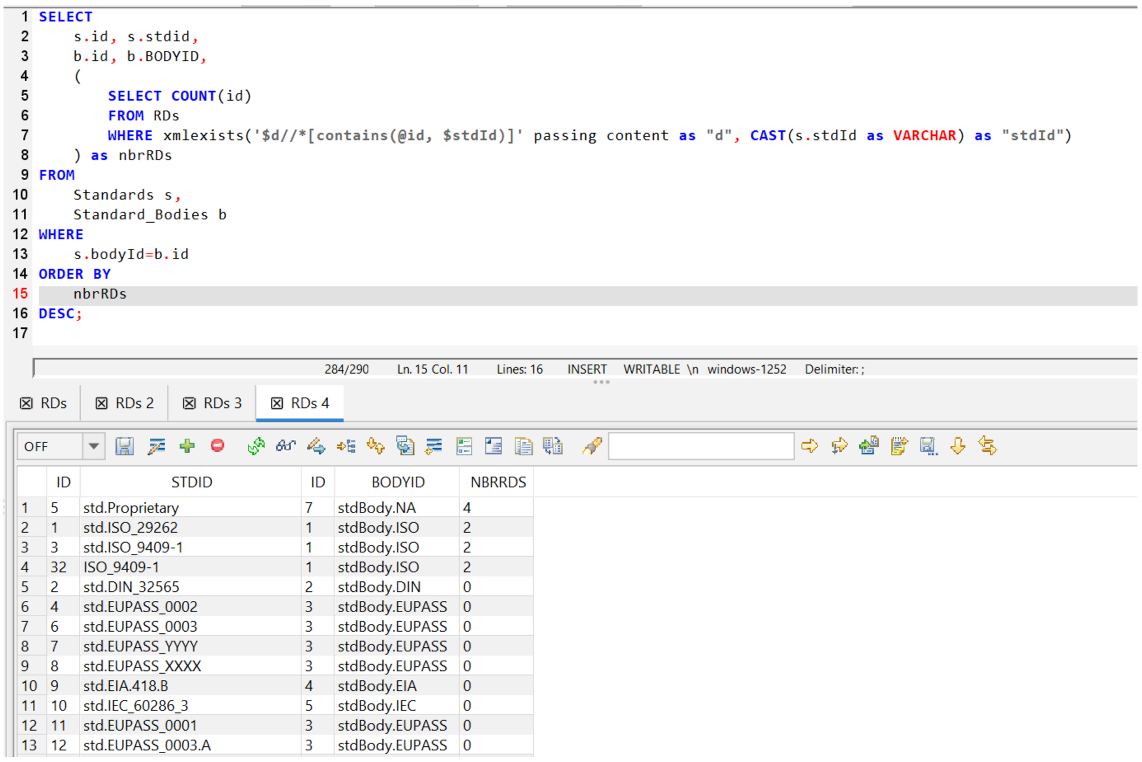
```
http://localhost:7070/api/v1/standards
```

200

```
  },
  {
    "id": 5,
    "stdId": "std.Proprietary",
    "code": null,
    "name": "Proprietary interface",
    "description": "Proprietary interface. Specific to the application/implementation.",
    "url": null,
    "nbrRDs": 4,
    "body": {
      "id": 7,
      "bodyId": "stdBody.NA",
      "name": "Not Available",
      "url": null,
      "description": "Not Available or unnamed Standardisation Body"
    }
  },
  {
    "id": 37,
    "stdId": "std.TUTuF.0001",
    "code": null,
    "name": "TUT uF - Cell Interface",
    "description": "Cell Interface between parallel TUT microFactory cells",
    "url": "http://www.tut.fi/microfactory",
    "nbrRDs": 0,
    "body": {
      "id": 8,
```

Figure 24. *A result of a list of Interface Standards*

To obtain the result shown in Figure 24, we utilized the query provided in section 5.5.3, which is an escalated version of the SQL query in section 5.5.2. This query exemplifies the potential of constructing complex SQL and XPATH/XQUERY statements to generate intricate results from a database design that leverages the XML datatype offered by IBM DB2.



```
SELECT
    s.id, s.stdid,
    b.id, b.BODYID,
    (
        SELECT COUNT(id)
        FROM RDs
        WHERE xmlexists('$d//*[contains(@id, $stdId)]' passing content as "d", CAST(s.stdId as VARCHAR) as "stdId")
    ) as nbrRDs
FROM
    Standards s,
    Standard_Bodies b
WHERE
    s.bodyId=b.id
ORDER BY
    nbrRDs
DESC;
```

	ID	STDID	ID	BODYID	NBRRDS
1	5	std.Proprietary	7	stdBody.NA	4
2	1	std.ISO_29262	1	stdBody.ISO	2
3	3	std.ISO_9409-1	1	stdBody.ISO	2
4	32	ISO_9409-1	1	stdBody.ISO	2
5	2	std.DIN_32565	2	stdBody.DIN	0
6	4	std.EUPASS_0002	3	stdBody.EUPASS	0
7	6	std.EUPASS_0003	3	stdBody.EUPASS	0
8	7	std.EUPASS_YYYY	3	stdBody.EUPASS	0
9	8	std.EUPASS_XXXX	3	stdBody.EUPASS	0
10	9	std.EIA.418.B	4	stdBody.EIA	0
11	10	std.IEC_60286_3	5	stdBody.IEC	0
12	11	std.EUPASS_0001	3	stdBody.EUPASS	0
13	12	std.EUPASS_0003.A	3	stdBody.EUPASS	0

Figure 25. *A result of complex SQL*

Furthermore, the Manager or Admin user mentioned in Table 1 can update the overall listing of all available interface specifications using the API documented in Figure 19. This allows the same user roles to modify the content of the Standard and Standard_Body tables without needing to directly interact with the database from a third-party client.

6.3 Concerns

Operating systems including Windows, Linux, and Macintosh can all be used to install IBM DB2. However, to facilitate development and testing, Docker is often used to quickly set up and run the database. Despite its convenience, the Docker version of DB2 has several limitations. For instance, it imposes a cap on memory allocation, which may negatively impact database performance, especially when working with large data sets. Additionally, the maximum disk space that can be allocated to the container is limited, thereby limiting the amount of data that can be stored. While the Docker version of DB2 offers scalability options, it may not be suitable for larger enterprises with high-traffic production workloads due to the inherent limitations of Docker containers.

In typical relational databases, backing up a database is usually accomplished by exporting the entire table into a SQL file. Database replication or migration involves running the exported SQL file on the newly installed database, and it is even possible to migrate to another relational database application by simply changing the suitable data types. However, XML is a unique data type in DB2, and backing up tables containing XML columns is not as simple as it may seem. A proper backup command must be followed, as per the IBM documentation.

In addition to these limitations, minor problems have been observed when running DB2 on Docker. For example, the credentials for JDBC connection are bound to the Linux user in the container, and the password is reset every 90 days without notification. It is also unclear how to set up the database with multiple users in the container. These issues could potentially hinder the effective and efficient use of DB2 in a Docker environment. Despite IBM's comprehensive documentation on installing DB2, our attempts to install it on our test server did not yield the expected results as reported in the tutorial [38].

7. CONCLUSIONS

The collaboration between the developer and Tampere University resulted in a thesis that aimed to address the challenge of upgrading an outdated ResourceCatalogue project, with the goal of developing a more maintainable and robust underlying architecture. The research question posed in the thesis pertained to the effectiveness of a hybrid database as a tool for managing XML resources in projects with long lifespans.

The initial section of the thesis entailed an introduction to the central concern of managing XML files within a singular folder. This discussion commenced with an overview of alternative approaches to managing XML resources, including both native XML databases and hybrid databases. Subsequently, the thesis delved into the specifics of utilizing hybrid databases in the context of web development. The section expounded on the capabilities of hybrid databases and analyzed the potential advantages they could offer to existing applications.

The subsequent section of the thesis involved an examination of the range of hybrid databases available, with a focus on selecting the most appropriate solution for the specific application. The section began with a clear articulation of the goals that the refactoring process was intended to achieve. A systematic plan was then formulated to convert the existing architecture and implementation into a RESTful application. After a thorough evaluation of the available options, IBM DB2 was deemed the optimal foundation for the application, as it offered the necessary features to accomplish the stated objectives.

The implementation process involved the utilization of cutting-edge, resilient technologies that were selected with the long-term maintainability of the project in consideration. Upon completion, the research findings indicated that IBM DB2 represents a credible choice for a database in web application projects that involve the management of XML resources. A retrospective review of the project's goals was conducted, which confirmed that they were accomplished without encountering significant obstacles that would have impeded successful completion. Overall, the outcomes of the project were satisfactory.

REFERENCES

[1] Harold ER. Effective XML : 50 specific ways to improve your XML. 1st edition. Boston: Addison Wesley; 2004.

[2] Siltala N. Resource Description Web Service - Public [Internet]. resourcedescription.rd.tuni.fi. 2006 [cited 2022 Feb 4]. Available from: https://resourcedescription.rd.tuni.fi/resourcedesc/

[3] Technology Corporation I. How can XML be Used? [Internet]. www.ibm.com. 2023 [cited 2023 Apr 4]. Available from: https://www.ibm.com/docs/en/scbn?topic=overview-how-can-xml-be-used

[4] Siltala N. Resource Description Concept [Internet]. resourcedescription.rd.tuni.fi. 2006. Available from: https://resourcedescription.rd.tuni.fi/resourcedesc/public/aboutEmplacementConcept.jsp

[5] Siltala N. Formal Digital Description of Production Equipment Modules for supporting System Design and Deployment [Internet] [PhD (Tech.) Thesis]. [Tampere University]; 2016. Available from: https://urn.fi/URN:ISBN:978-952-15-3783-7

[6] Pfleeger, S. L., & Atlee, J. M. (2009). Software engineering: theory and practice. Pearson Education.

[7] Sommerville, I., & Sawyer, P. (2007). Software engineering. Pearson Education.

[8] Pales S. The ultimate guide to resource management [Internet]. Tempus Resource. Prosymmetry; 2017 [cited 2023 Jan 5]. Available from: https://www.prosymmetry.com/resources/know-your-resource-management-techniques/

[9] W3C. XML Core Working Group Public Page [Internet]. www.w3.org. 2017 [cited 2023 Feb 4]. Available from: https://www.w3.org/XML/Core/#Publications

[10] Elliotte Rusty Harold, W. Scott Means. XML in a Nutshell, 3rd Edition. O'Reilly Media, Inc; 2004.

[11] Advantages of XML [Internet]. Ibm.com. 2021 [cited 2023 Jan 6]. Available from: https://www.ibm.com/docs/en/i/7.3?topic=introduction-advantages-xml

[12] Uses of XML [Internet]. Ibm.com. 2021 [cited 2023 Jan 6]. Available from: https://www.ibm.com/docs/en/i/7.2?topic=introduction-uses-xml

[13] Fawcett J, Ayers D, Quin LRE. Beginning XML. 5th ed. Wrox Press; 2012.

[14] Miller S. What is XML used for? [Internet]. Codecademy News. 2021 [cited 2023 Jan 6]. Available from: https://www.codecademy.com/resources/blog/what-is-xml-used-for/

[15] Niederst J. Web Design in a nutshell: a desktop quick reference. Sebastopol, Calif: O'reilly; 2006.

[16] Chen W-J, Sammartino A, Goutev D, Hendricks F, Komi I, Wei M-P, et al. DB2 9 PureXML Guide. IBM.com/Redbooks; 2007.

[17] Mccreary D, Kelly A. Making sense of NoSQL: a guide for managers and the rest of us. Shelter Island, N.Y.: Manning; 2013.

[18] Chaudhri A, Rashid A, Zicari R, Fuller J. XML Data Management: Native XML and XML-Enabled Database Systems. Boston, Ma: Addison-Wesley; 2003.

[19] Siegel E, Retter A. eXist. O'Reilly Media, Inc.; 2014.

[20] Documentation [Internet]. eXist-db. 2014 [cited 2023 Jan 6]. Available from: http://exist-db.org/exist/apps/doc/

[21] Database Deployment [Internet]. eXist-db. 2014 [cited 2023 Jan 6]. Available from: http://exist-db.org/exist/apps/doc/deployment#run-stand-alone

[22] REST-Style Web API [Internet]. eXist-db. 2014 [cited 2023 Jan 6]. Available from: http://exist-db.org/exist/apps/doc/devguide_rest

[23] Palsetia N, Deepa G, Ahmed Khan F, Thilagam PS, R. Pais A. Securing native XML database-driven web applications from XQuery injection vulnerabilities. Journal of Systems and Software. 2016 Dec;122:93–109.

[24] Appelquist DK. XML and SQL: Developing Web Applications. Addison-Wesley Professional; 2001.

[25] Revesz P. Introduction to Databases From Biological to Spatio-Temporal. 1st ed. 2010. London: Springer London; 2010.

[26] Silberschatz A, Korth H, Sudarshan S. Database system concepts. 7th ed. McGraw-Hill Education; 2020.

[27] Nicola M, Kumar-Chatterjee P. DB2® pureXML® Cookbook: Master the Power of the IBM® Hybrid Data Server. 1st ed. IBM Press; 2009.

[28] Staken K. Introduction to Native XML Databases [Internet]. www.xml.com. 2001 [cited 2023 Jan 11]. Available from: https://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[29] Sadanala Chowdhary G. JSP Architecture [Internet]. GeeksforGeeks. 2021. Available from: https://www.geeksforgeeks.org/jsp-architecture/

[30] Apache Tomcat Project. Apache Tomcat® - Welcome! [Internet]. Apache Tomcat. 2019 [cited 2023 Jan 11]. Available from: https://tomcat.apache.org/

[31] Amazon Web Services. Three-Tier Architecture Overview - AWS Serverless Multi-Tier Architectures with Amazon API Gateway and AWS Lambda [Internet]. docs.aws.amazon.com. 2021. Available from: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html

[32] Ibm.com. Product overviews [Internet]. DB2 Documentation. 2022 [cited 2023 Jan 21]. Available from: https://www.ibm.com/docs/en/db2/11.5?topic=product-overviews

[33] Oracle. XML DB Developer's Guide [Internet]. docs.oracle.com. 2016 [cited 2023 Mar 2]. Available from: https://docs.oracle.com/database/121/ADXDB/xdb01int.htm#ADXDB3994

[34] MikeRayMSFT. XML data (SQL Server) - SQL Server [Internet]. learn.microsoft.com. 2023 [cited 2023 Mar 5]. Available from: https://learn.microsoft.com/en-us/sql/relational-databases/xml/xml-data-sql-server?view=sql-server-ver16

[35] Wooldridge B. HikariCP It’s Faster.Hi·ka·ri [hi·ka·'lē] (Origin: Japanese): light; ray. [Internet]. GitHub. 2022 [cited 2023 Mar 10]. Available from: https://github.com/brettwooldridge/HikariCP

[36] Siltala N. Software Requirements Specification for Resource Catalogue Platform V1.0. Tampere University; 2017.

[37] Siltala N. Summer trainee on web programming. [PowerPoint presentation]. TAU / Faculty of Engineering and Natural Sciences (ENS). [updated 2021 March 15; cited 2023 Apr 19]

[38] IBM. Installing Db2 servers by using the Db2 Setup wizard (Linux and UNIX) [Internet]. www.ibm.com. 2023 [cited 2023 Apr 19]. Available from: https://www.ibm.com/docs/en/db2/11.5?topic=idds-installing-db2-servers-by-using-db2-setup-wizard-linux-unix