UNIVERSITÄT ZU LÜBECK

Information Systems

CS4130-KP06

Prof. Dr. Sylvia Melzer

SoSe2026

Building Information Systems with JavaScript

Information Systems

Building Information Systems with JavaScript

- From structured research data to interactive systems
- Bridging data transformation and user interaction
- Focus on implementation and system integration
- Based on Humanities use cases

From Data Model to Information System

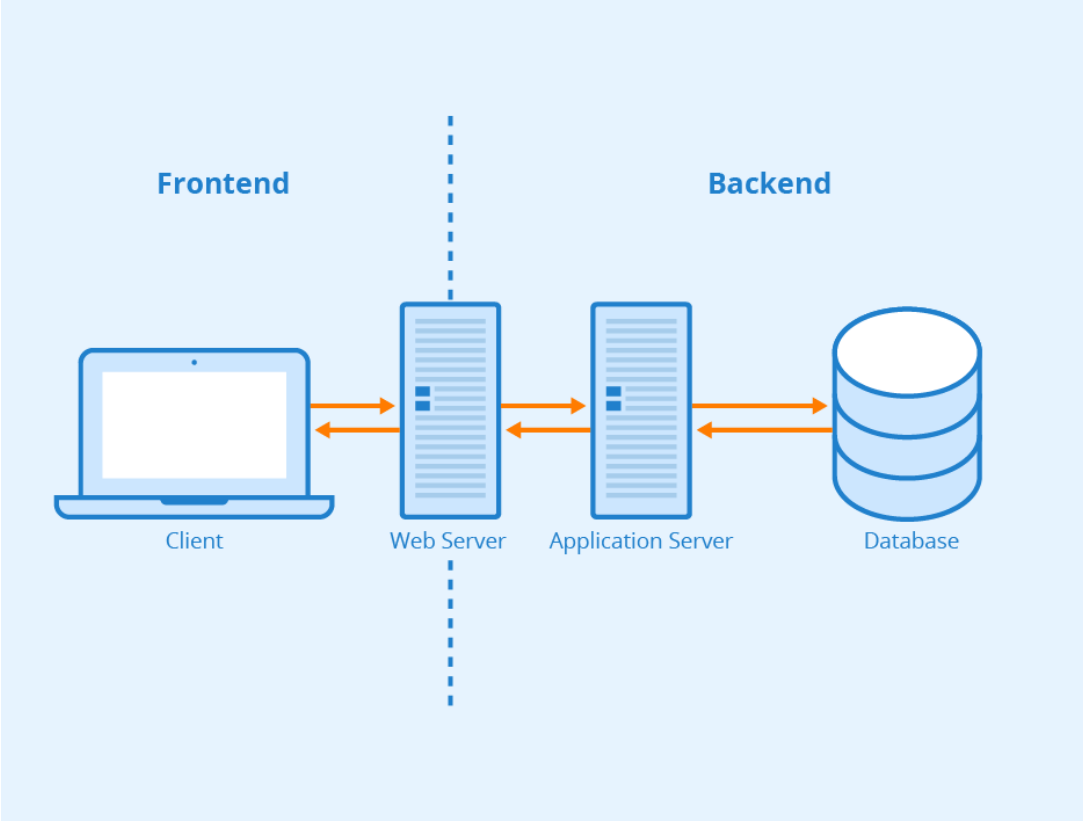
- Data has been modeled (e.g., TEI, JSON, CSV)
- Data has been transformed (e.g., DBoD: XSLT, Python)
- Data is stored in repositories (e.g., RDR)
- But: no direct interaction for users
- No querying, filtering, or exploration
- No interface for research questions

→The information system layer is missing

From Data to System

- Data pipeline produces structured outputs (CSV/HML)
- These outputs are not yet usable systems
- A system requires interaction and access
- Users need filtering, querying, visualization
- Systems must respond dynamically to input
- Transformation ≠ system

→ A new layer must be introduced



https://medium.com/coders-capsule/getting-started-with-backend-development-8ce55585e860

Why Building Information Systems?

- Research data alone is not directly usable
- Large datasets require structured access
- Researchers need filtering and exploration
- Generic repository interfaces are often insufficient
- Project-specific interfaces support research questions
- Scalability becomes critical for growing datasets

→ **Information systems bridge data and usage**

Technical View on an Information System

- A system consists of interconnected components
- Data layer: storage and structured datasets
- Logic layer: processing and querying
- Interface layer: user interaction
- Communication via APIs
- Real-time response to user input

→Focus today: implementation with JavaScript

What You Will Learn

- Understand how information systems emerge from research data pipelines
- Learn how JavaScript connects repositories, APIs, and user interfaces
- Compare frontend-only and database-driven architectures
- Build interactive viewers for structured research data
- Understand the role of repositories such as Zenodo and RDR@UHH
- Evaluate scalability challenges in web-based systems
- Reflect on architectural decisions for research infrastructures

Architecture

- Browser acts as the user interface
- Backend provides logic and data access
- Database stores structured datasets
- Communication via HTTP requests
- Data exchanged in JSON format
- Clear separation of concerns

→ **Standard architecture of web-based systems**

Workflow Mapping

- EpiDoc → transformed into CSV (existing pipeline)
- CSV represents structured analytical data
- JavaScript backend exposes the data
- Frontend requests and visualizes the data
- Users interact via filtering and search
- System responds dynamically

Role of JavaScript

- JavaScript runs in the browser (frontend)
- JavaScript runs on the server (Node.js)
- Enables full-stack development
- Handles asynchronous data loading
- Processes and transforms JSON data
- Connects UI with backend logic

JavaScript - Features

- Influences from other programming languages
  - Notation similar to C/C++/Java
  - But little relation to Java
- Interpreted
- Dynamically typed
- Specific object-oriented concepts
- Interpreter integrated into web browsers
- Closely linked to HTML
  - Access to elements of the displayed document via the DOM API

Example: Web Application using JavaScript

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>JavaScript Example</title>
5  <script type="text/javascript">
6  var number = Math.floor(Math.random()*6)+1;
7  function check() {
8  var guessed = document.GuessForm.inputfield.value;
9  if(guessed==number){
10 alert("Congratulations! The dice value is "
11 + number + ".");
12 } else {
13 if(number<guessed) alert("My number is smaller!");
14 else alert("My number is bigger!");
15 I have thrown a dice.
16 Please guess the
17 number:
18 }
19 }
20 </script>
21 </head>
22 <body>
23 I have thrown a dice. Please guess the number:
24 <form name="GuessForm">
25 <input type="number" name="inputfield"
26 size="1" min="1" max="6"/>
27 <input type="button" value="Guess!" onClick="check()"/>
28 Guess!
29 </form>
30 </body>
31 </html>
```

I have thrown a dice. Please guess the number:

Guess! Guess!

Embedding JavaScript in HTML

1. As a script section within an HTML document:

```
<script type="text/javascript">
… // your javascript code goes here…
</script>
```

2. Within HTML tags (to define handlers that are triggered by specific events):

a. Defining the event as an attribute: The attribute value contains a sequence of instructions that is executed when the event occurs:

```
<input type="button" value="…" onClick="checkGuess()"/>
```

b. Specifying a sequence of instructions in an anchor element (instead of a URL) using javascript:, which is executed when clicked:

```
<a href="javascript:checkGuess()">…</a>
```

Embedding JavaScript in HTML

3. In a separate file:

```
<script src="file.js" type="text/javascript"></script>
```

- The <script> element can be used multiple times
- JavaScript code executes immediately when the browser loads the document
  - Function definitions and event handlers do not execute immediately
- Multiple statements can be specified in HTML attributes,cseparated by semicolons (;), but this can be confusing
- Reduced loading time by placing JavaScript sections at the end of the document
- Separate file (exclusively) containing JavaScript code: file with the .js file

Data Flow in an Information System

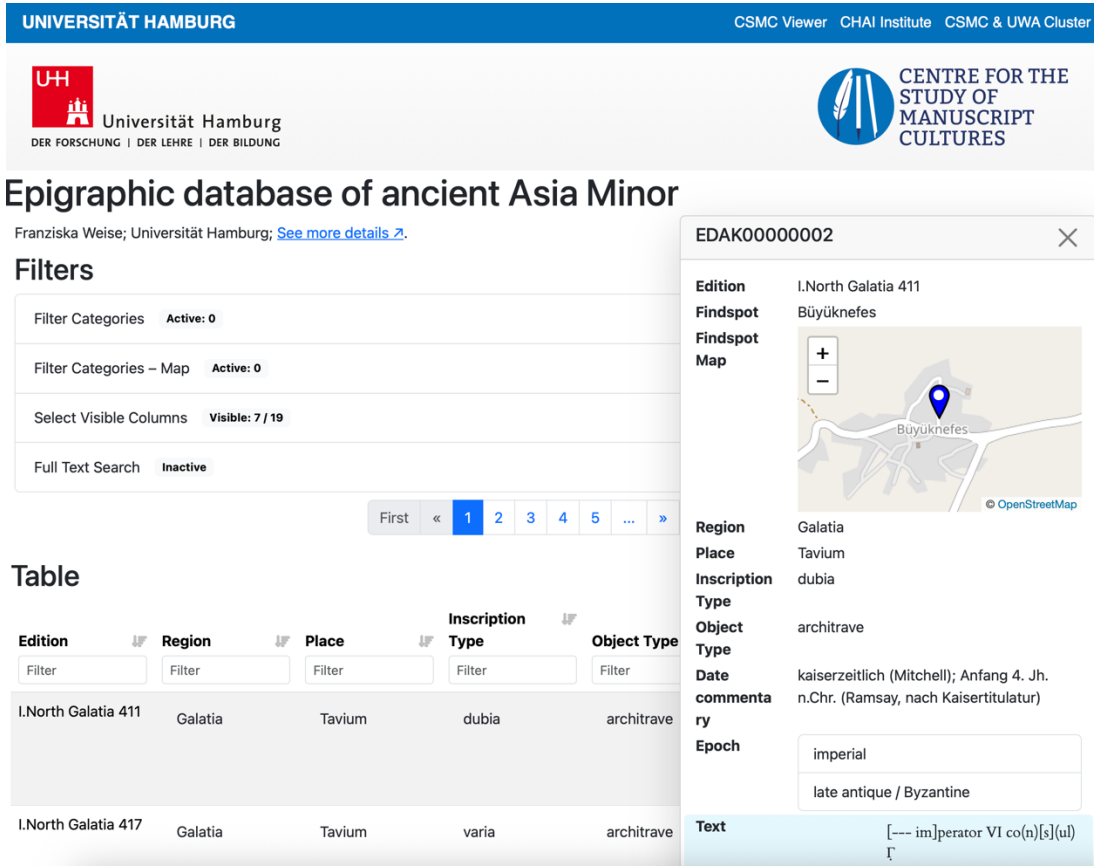
- User performs an action (search/filter)
- Frontend sends request to API
- Backend processes the request
- Data is filtered or transformed
- Response is sent as JSON
- Frontend renders updated view

Frontend: What Does a Viewer Do?

- Displays structured data
- Allows filtering and searching
- Supports sorting and navigation
- Provides detail views
- Enables comparison of records
- Responds to user interaction
- Helps to analyze (research) data

Core Features of a Data Viewer

- Table-based representation of data
- Column-based filtering
- Row-level detail inspection
- Dynamic updates without reload
- Search across multiple fields
- Sorting and ordering



https://www.fis.uni-hamburg.de/en/publikationen/detail.html?id=27acbbfc-4deb-429d-86d7-7d98afa34c2e

Fetching Data (Concept)

- Frontend requests data via HTTP
- Uses Fetch API
- Asynchronous data loading
- No page reload required
- Data received as JSON
- Enables dynamic interfaces

Rendering Data

- Data must be transformed into HTML
- Tables are common representations
- Each row = one analytical unit
- Columns represent attributes
- Rendering updates dynamically
- Based on user interaction

→ **Data becomes visible and usable**

Backend & API

- Why an API?
- Separates frontend and backend logic
- Provides structured data access
- Enables reuse across applications
- Supports multiple clients
- Standardizes communication
- Improves scalability

Repositories as System Components

- Store and publish research data
- Provide metadata and identifiers
- Ensure FAIR compliance
- Limited interaction capabilities
- No advanced filtering or analysis
- No custom interfaces

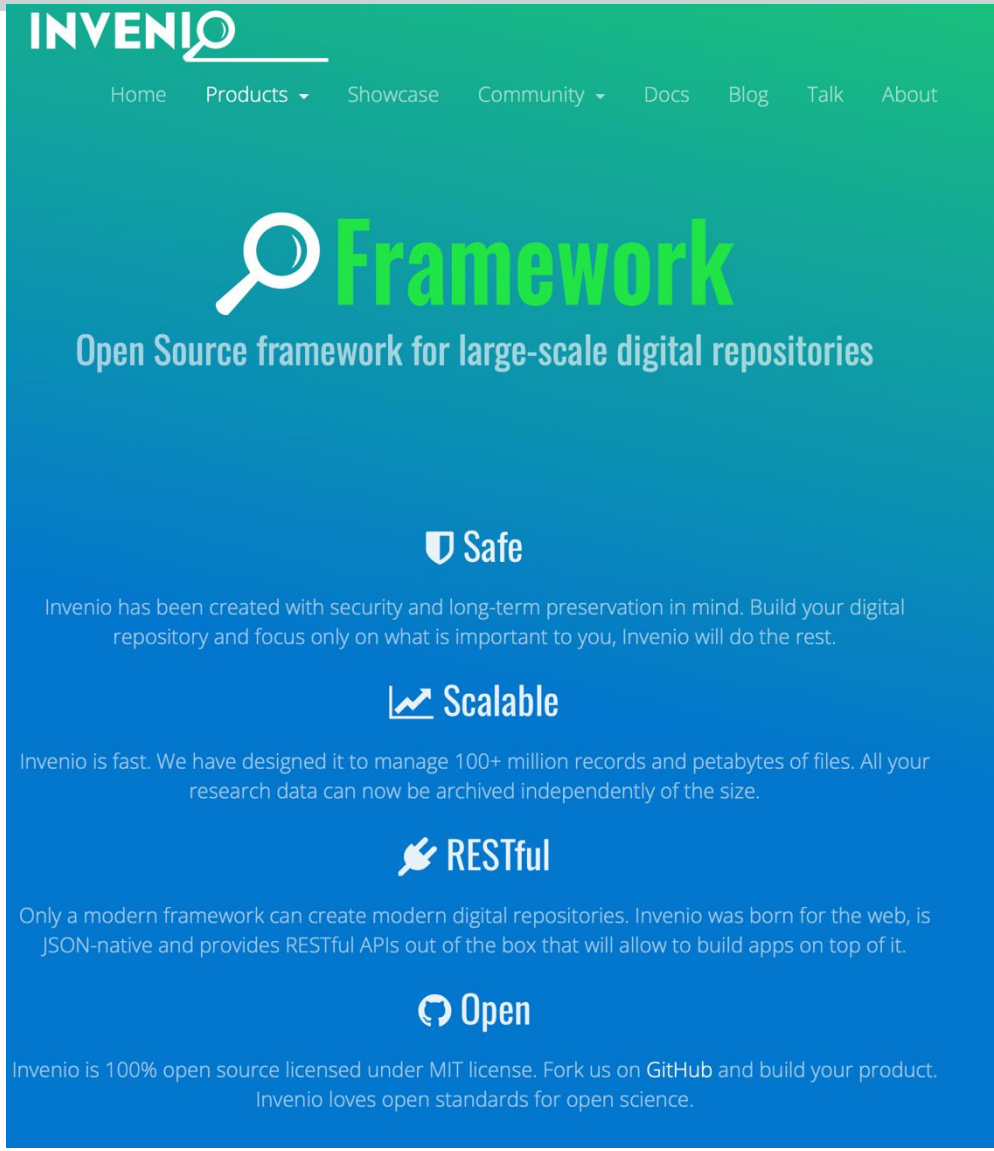
→ **Not full information systems**

Metadata Standards in RDRs

- DOI assignment and metadata standards are essential for research data archiving
- Key standards include DataCite, MARC, MODS, and Dublin Core
- These standards enable identification, description, and interoperability of data
- Different repositories implement these standards in different ways. Examples: InvenioRDM, RDR@UHH, and Zenodo
- Implementation depends on goals such as usability, interoperability, and community needs
- Comparing implementations reveals strengths and limitations for data management and preservation

Invenio Framework

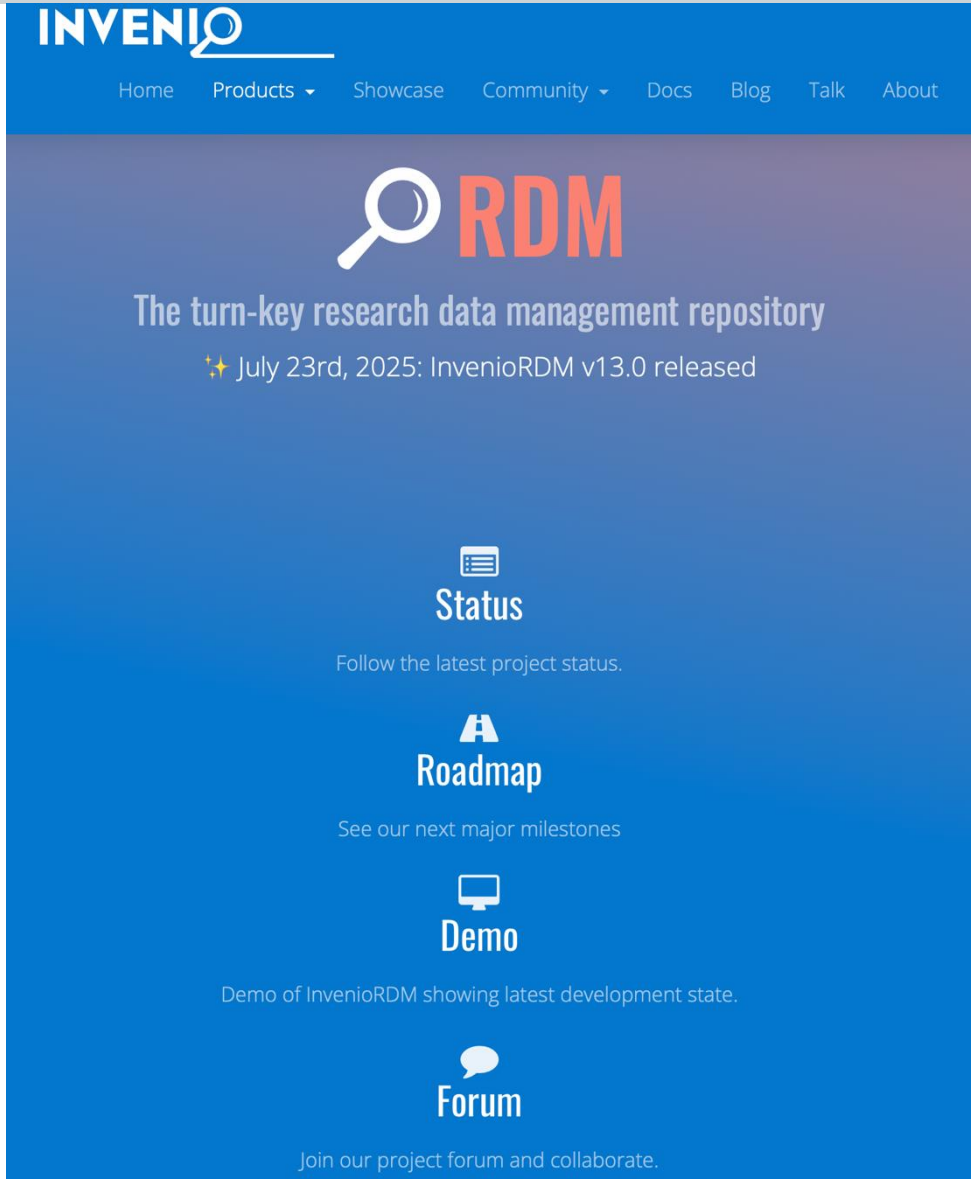
- Open-source platform for building digital repositories
- Originally developed by CERN (Conseil européene pour la recherche nucléaire)
- Provides a modular and extensible architecture
- Allows customization for different disciplines and use cases
- Supports access control and user management
- Includes file storage and search functionalities
- Designed for scalable and robust RDM systems
- is JSON-native and provides RESTful APIs out of the box that will allow to build apps on top of it



https://inveniosoftware.org/products/framework/

InvenioRDM

- Open-source repository platform specifically designed for RDM
- Developed for institutions and organizations worldwide
- Highly flexible and extensible architecture
- Supports multiple metadata schemas (e.g., DataCite Metadata Schema)
- Enables customization for project- or domain-specific needs
- Ensures secure storage and long-term archiving of research data
- Promotes findability and interoperability across systems

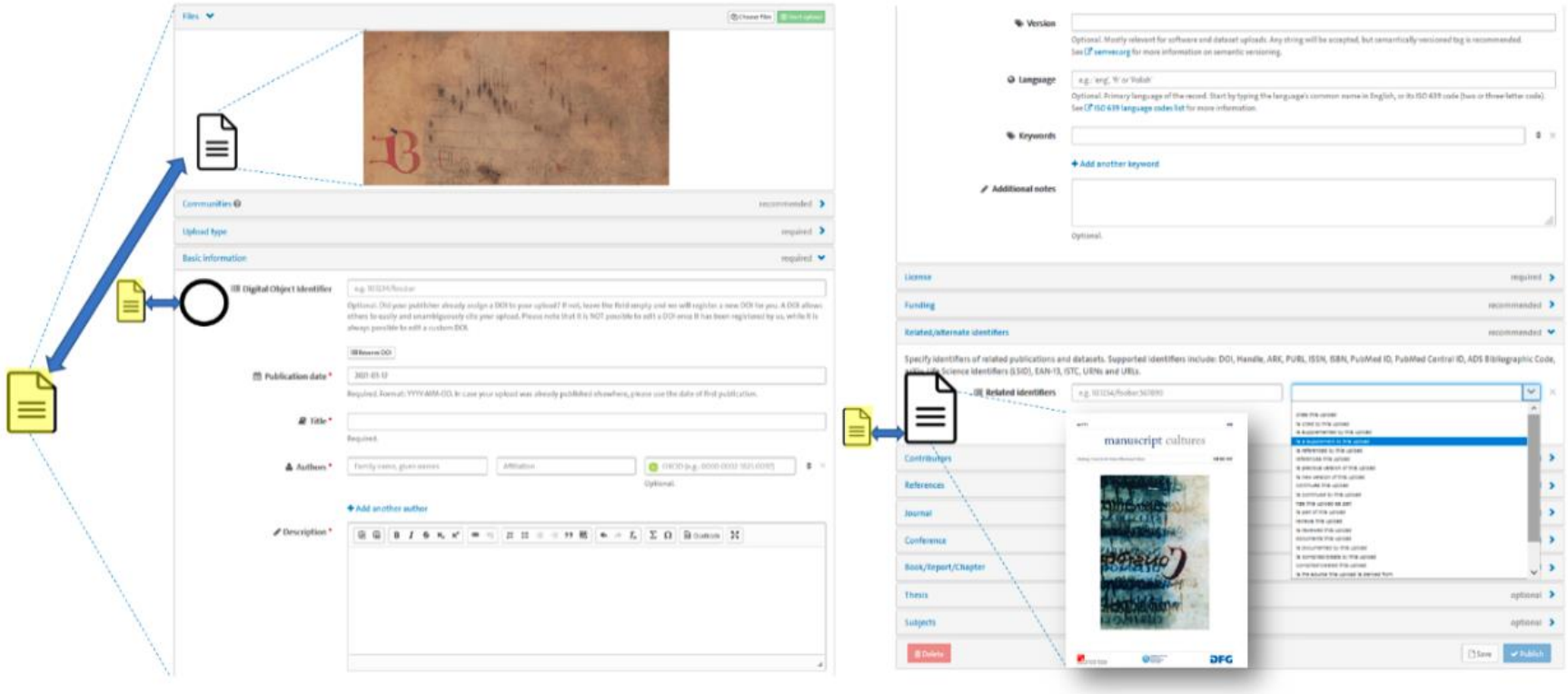


https://inveniosoftware.org/products/rdm/

RDR@UHH

- Institutional repository for long-term storage and sharing of research data at the University of Hamburg
- Access restricted to UHH researchers for publishing; open access data available to all users
- Supports diverse data types (e.g., texts, measurements, audiovisual data, software) using DataCite metadata
- Enables linking of related datasets (e.g., references, supplements, part–whole relationships)
- Provides enhanced search across metadata and linked data, improving discoverability
- Offers export formats (e.g., BibTeX, JSON, Dublin Core, MARCXML) for interoperability and reuse
- Includes generic viewer and DBoD approach for flexible, project-specific data exploration (**NEW since 2026**)

RDR@UHH (Upload)



Upload research data at the RDR. Yellow document symbol: RDR research data; white document symbol: research data, e.g., publications, measurements and other resources; white dot: DOI; blue arrow: linking of data.

RDR@UHH (Linking)

link to supplementing document

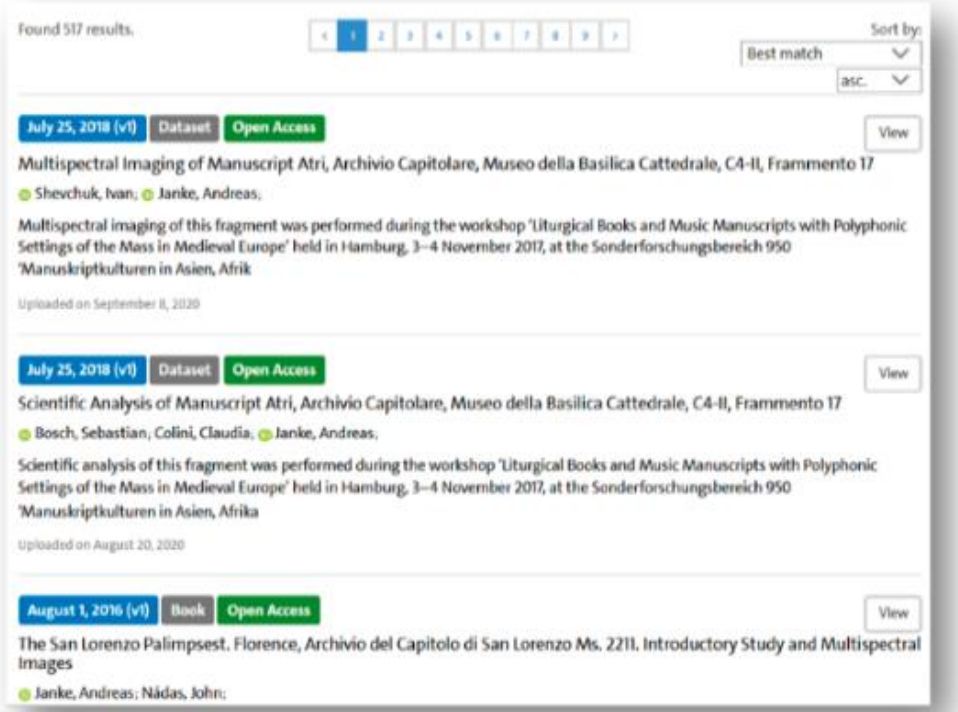
https://doi.org/10.25592/uhhfdm.1472

RDR@UHH (Search results)

RDR

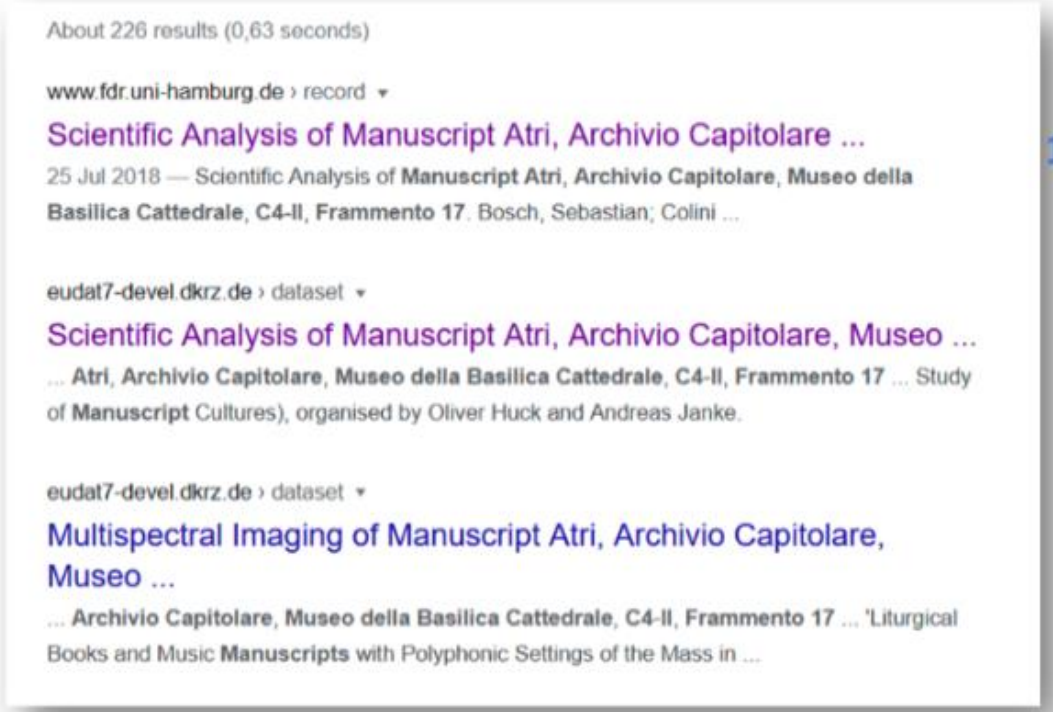
1472

1430



Google

1430



Google has not found the record 1472.

Zenodo

- Open-access repository built on Invenio and hosted at CERN
- Developed within the OpenAIRE initiative funded by the European Commission
- Available to researchers worldwide for publishing data, software, and publications
- Automatically assigns DOIs to all uploads, improving citation and discoverability
- Supports linking between related datasets (similar to RDR@UHH)
- Offers a wide range of export formats (e.g., JSON, DataCite, Dublin Core, MARCXML, BibTeX, GeoJSON)
- Designed for broad dissemination and interoperability across international research communities

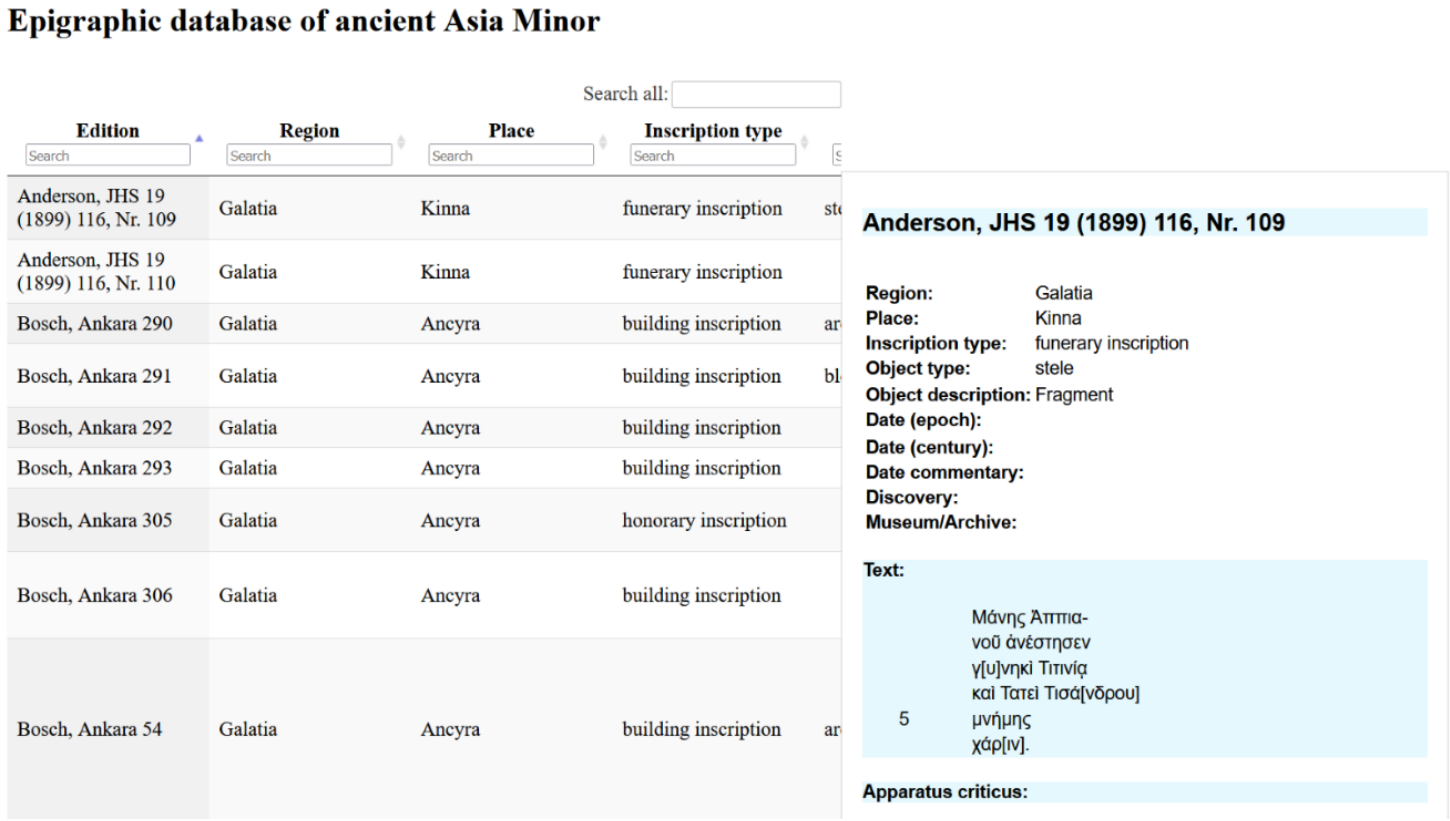
Why Repositories Are Not Enough

- Provide storage, not interaction
- Limited filtering and querying
- No custom analysis tools
- Generic interfaces only
- Not tailored to research questions
- No project-specific views

DataTable Approach

- Simple frontend-based table rendering
- Uses libraries like DataTables
- Data loaded entirely into browser
- Supports filtering and sorting
- Easy to implement
- Works well for small datasets

→**Good for rapid prototyping**



Epigraphic database of ancient Asia Minor

Search all:

Edition	Region	Place	Inscription type	
Anderson, JHS 19 (1899) 116, Nr. 109	Galatia	Kinna	funerary inscription	st
Anderson, JHS 19 (1899) 116, Nr. 110	Galatia	Kinna	funerary inscription	
Bosch, Ankara 290	Galatia	Ancyra	building inscription	ar
Bosch, Ankara 291	Galatia	Ancyra	building inscription	bl
Bosch, Ankara 292	Galatia	Ancyra	building inscription	
Bosch, Ankara 293	Galatia	Ancyra	building inscription	
Bosch, Ankara 305	Galatia	Ancyra	honorary inscription	
Bosch, Ankara 306	Galatia	Ancyra	building inscription	
Bosch, Ankara 54	Galatia	Ancyra	building inscription	ar

Anderson, JHS 19 (1899) 116, Nr. 109

Region: Galatia
Place: Kinna
Inscription type: funerary inscription
Object type: stele
Object description: Fragment
Date (epoch):
Date (century):
Date commentary:
Discovery:
Museum/Archive:

Text:

Μάνης Ἀππια-
νοῦ ἀνέστησεν
γ[υ]νηκὶ Τιτινίᾳ
καὶ Τατεὶ Τισά[νδρου]
μνήμης
χάρ[ιν].

Apparatus criticus:

HTML

- For DataTables to be able to enhance an HTML table, the table must be valid, well formatted HTML, with a header (thead) and a single body (tbody)
- An optional footer (tfoot) can also be used

HTML

```html
<table id="myTable" class="display">
    <thead>
        <tr>
            <th>Column 1</th>
            <th>Column 2</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Row 1 Data 1</td>
            <td>Row 1 Data 2</td>
        </tr>
        <tr>
            <td>Row 2 Data 1</td>
            <td>Row 2 Data 2</td>
        </tr>
    </tbody>
</table>
```


https://datatables.net/manual/installation

DataTables (Javascript library)

- DataTables is a powerful Javascript library for adding interaction features to HTML tables, and while simplicity is a core design principle for the project as a whole
- You need only include two additional files in your page:
  - The DataTables Javascript file
  - The DataTables CSS file

```HTML
<link rel="stylesheet" href="/DataTables/datatables.css" />

<script src="/DataTables/datatables.js"></script>
```

- DataTables has only one library dependency →jQuery
- JQuery: The most widely used JavaScript library, used on over 70% of websites
- DataTables uses jQuery as a utility library, allowing the venerable library to deal with many of the complexities of event handling and DOM manipulation

https://datatables.net/manual/installation

DataTables

With jQuery

```
1  $(document).ready( function () {
2      $('#myTable').DataTable();
3  } );
```

w/o jQuery

```
1  let table = new DataTable('#myTable', {
2      // options
3  });
```


https://datatables.net/manual/installation

HTML

```html
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>DataTables Beispiel</title>
    <link rel="stylesheet" href="https://cdn.datatables.net/2.3.7/css/dataTables.dataTables.css">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="https://cdn.datatables.net/2.3.7/js/dataTables.js"></script>
    <script defer src="app.js"></script>
</head>
<body>

<h2>Benutzerliste</h2>
<table id="example" class="display">
    <thead>
        <tr>
            <th>Name</th>
            <th>Position</th>
            <th>Ort</th>
            <th>Alter</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Max Mustermann</td>
            <td>Entwickler</td>
            <td>Hamburg</td>
            <td>30</td>
        </tr>
```

HTML

```
<table id="example" class="display">
    <thead>
        <tr>
            <th>Name</th>
            <th>Position</th>
            <th>Ort</th>
            <th>Alter</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Max Mustermann</td>
            <td>Entwickler</td>
            <td>Hamburg</td>
            <td>30</td>
        </tr>
        <tr>
            <td>Anna Schmidt</td>
            <td>Designer</td>
            <td>Berlin</td>
            <td>27</td>
        </tr>
        <tr>
            <td>John Doe</td>
            <td>Manager</td>
            <td>München</td>
            <td>40</td>
        </tr>
    </tbody>
</table>
</body>
</html>
```

Benutzerliste

Name	Position	Ort	Alter
Max Mustermann	Entwickler	Hamburg	30
Anna Schmidt	Designer	Berlin	27
John Doe	Manager	München	40

JS

```
$(document).ready(function () {
    $('#example').DataTable({
        paging: true,
        searching: true,
        ordering: true,
        pageLength: 5,
        language: {
            search: "Suche:",
            lengthMenu: "Zeige _MENU_ Einträge",
            info: "Zeige _START_ bis _END_ von _TOTAL_ Einträgen"
        }
    });
});
```

Benutzerliste

Zeige [ ] Einträge Suche: [ ]

Name	Position	Ort	Alter
Anna Schmidt	Designer	Berlin	27
John Doe	Manager	München	40
Max Mustermann	Entwickler	Hamburg	30

Zeige 1 bis 3 von 3 Einträgen « ‹ 1 › »

HTML + JS w/o jQuery

HTML

```html
<table id="example">
    <thead>
        <tr>
            <th>Name</th>
            <th>Position</th>
            <th>Ort</th>
            <th>Alter</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Max Mustermann</td>
            <td>Entwickler</td>
            <td>Hamburg</td>
            <td>30</td>
        </tr>
    </tbody>
</table>
```

JS

```js
new DataTable('#example', {
    paging: true,
    searching: true,
    ordering: true
});
```

Implementation Process

- Step 1: Retrieve data via API
- Step 2: Transform into JSON
- Step 3: Process in backend
- Step 4: Serve via API
- Step 5: Render in viewer
- Step 6: Enable interaction

When Does DataTables Stop Working?

- Small datasets → responsive interaction
- Medium datasets → slower filtering
- Large datasets → browser bottlenecks
- Full dataset loaded into memory
- Rendering becomes inefficient
- User experience degrades

→ Need database-driven architectures

From Frontend Tables to Distributed Systems

- Frontend-only viewers are easy to build
- But scalability becomes a challenge
- Large datasets exceed browser capabilities
- Offline access may be required
- Multiple users need synchronization
- Distributed architectures solve these problems

→ **Approach: PouchDB and CouchDB**

PouchDB

- PouchDB is an in-browser database that allows applications to save data locally, so that users can enjoy all the features of an app even when they're offline
- The data is synchronized between clients, so users can stay up-to-date wherever they go
- PouchDB also runs in Node.js and can be used as a direct interface to CouchDB-compatible servers
- The API works the same in every environment, so you can spend less time worrying about browser differences, and more time writing clean, consistent code
- PouchDB is a free open-source project, written in JavaScript

https://pouchdb.com/learn.html

PouchDB

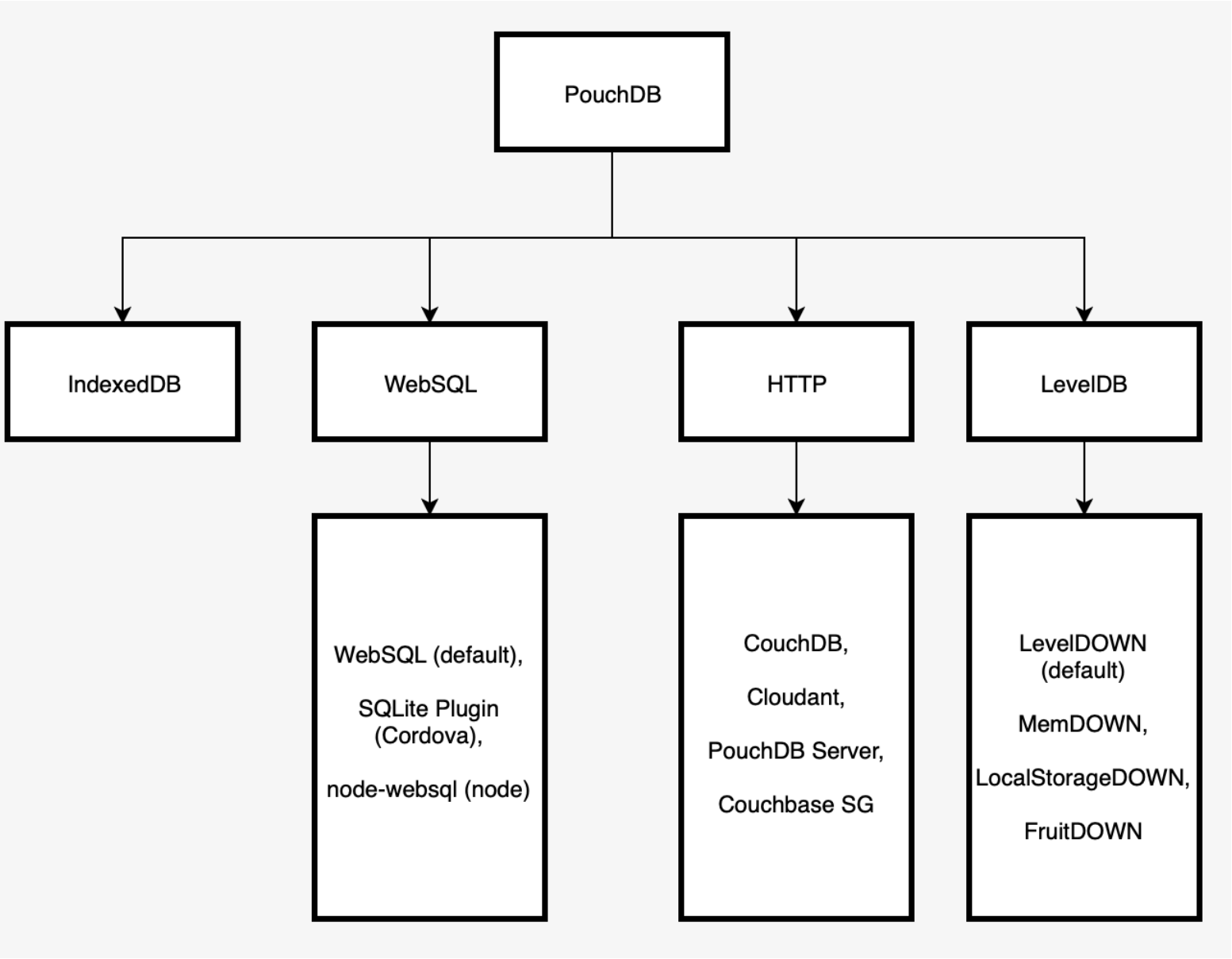
- PouchDB supports all modern browsers, using IndexedDB under the hood and falling back to WebSQL where IndexedDB isn't supported. It is fully tested and supported in:
  - Firefox 29+ (Including Firefox OS and Firefox for Android)
  - Chrome 30+
  - Safari 5+
  - Internet Explorer 10+
  - Opera 21+
  - Android 4.0+
  - iOS 7.1+
  - Windows Phone 8+

IndexedDB API

- IndexedDB is a low-level API for storing large amounts of structured data on the client side, including files and blobs
- This API uses indexes to enable high-performance searches of this data
- While Web Storage is useful for storing smaller amounts of data, it is less suitable for storing larger amounts of structured data → IndexedDB offers a solution for this
- IndexedDB is a transactional database system similar to an SQL-based relational database management system (RDBMS)
- Unlike SQL-based RDBMSs, which use fixed-column tables, IndexedDB is a JavaScript-based object-oriented database
- IndexedDB allows you to store and retrieve objects that are indexed by a key

https://developer.mozilla.org/de/docs/Web/API/IndexedDB_API

PouchDB

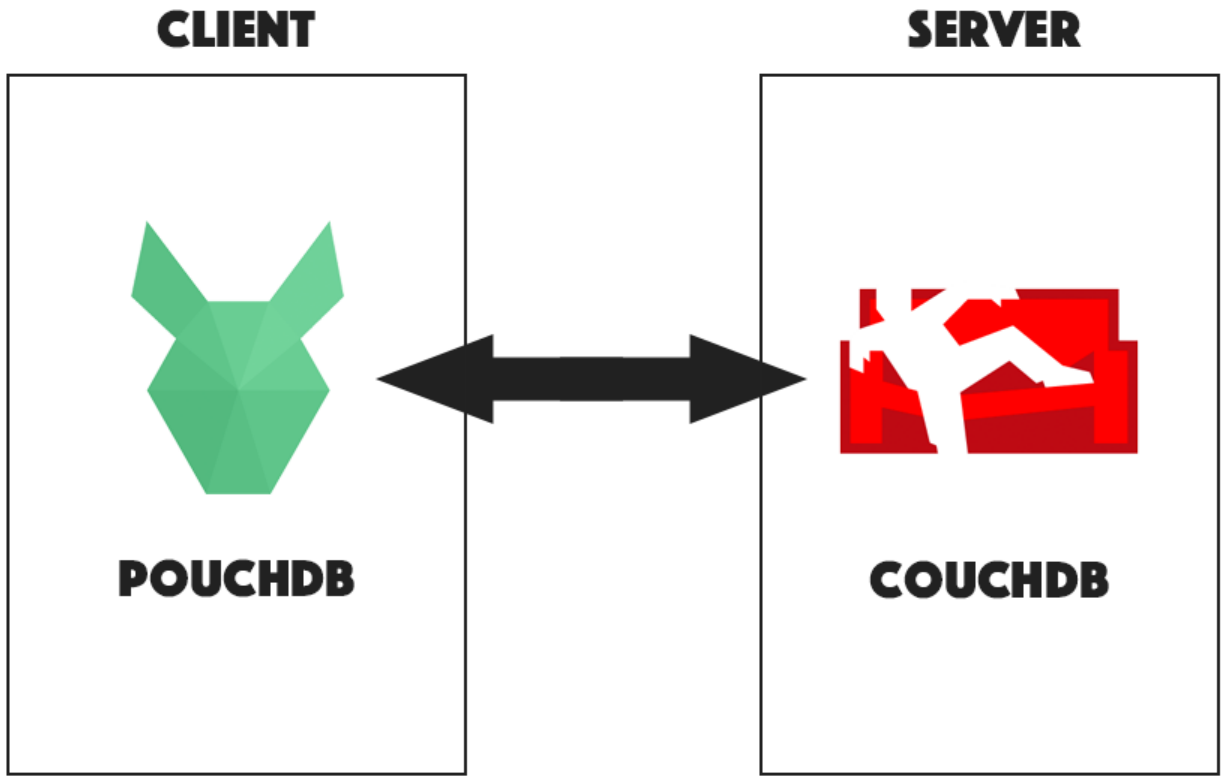

PouchDB
IndexedDB
WebSQL
HTTP
LevelDB
WebSQL (default),
SQLite Plugin (Cordova),
node-websql (node)
CouchDB,
Cloudant,
PouchDB Server,
Couchbase SG
LevelDOWN (default)
MemDOWN,
LocalStorageDOWN,
FruitDOWN

CouchDB

- Apache CouchDB is an open-source document-oriented NoSQL database, implemented in Erlang
- CouchDB uses multiple formats and protocols to store, transfer, and process its data
- It uses JSON to store data, JavaScript as its query language using MapReduce, and HTTP for an API.

https://en.wikipedia.org/wiki/Apache_CouchDB

A Database Built to Embrace Conflict



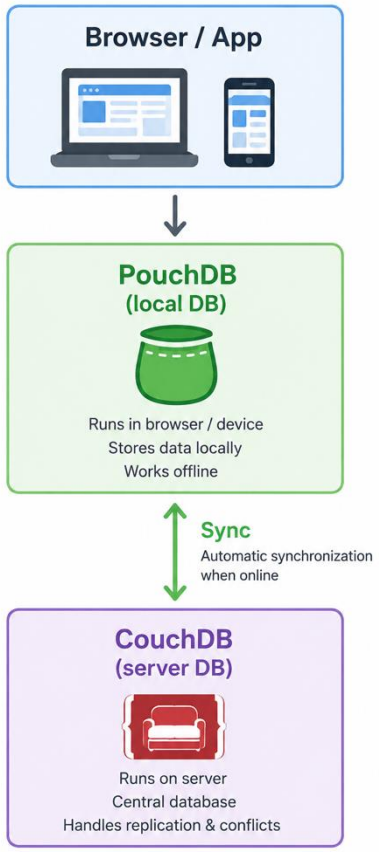
https://medium.com/%40baalaaraman/the-only-database-built-to-embrace-conflict-a-deep-dive-into-couchdb-pouchdb-bdfbfe48e2bb

CouchDB – Aim

- Traditional databases try to avoid conflicts using locks and strict consistency
- CouchDB is designed to embrace conflicts and resolve them later
- Ideal for:
  - Offline-first apps
  - Distributed systems
  - Multi-device synchronization
  - Unreliable network environments

CouchDB + PouchDB Workflow

1. User works locally in browser (PouchDB)
2. Data stored offline
3. Internet reconnects
4. PouchDB syncs with CouchDB
5. Conflicts handled through revision system



CouchDB + PouchDB Workflow

Advantages

- Excellent offline support
- Seamless synchronization
- Fault-tolerant architecture
- Scales across distributed systems
- Easy JSON-based development

Challenges

- Conflict resolution complexity
- Eventual consistency may confuse developers
- Not ideal for:
- Strong transactional systems
- Complex relational queries

Build Viewer with PouchDB (HTML)

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.11.5/css/jquery.dataTables.css">
  <script type="text/javascript" charset="utf8" src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script type="text/javascript" charset="utf8" src="https://cdn.datatables.net/1.11.5/js/jquery.dataTables.js"></script>
<title>EDAK.CSV-Daten mit PouchDB darstellen</title>
<style>
  tfoot {
    display: table-header-group;
  }
</style>
</head>
<body>
  <h1>EDAK 2</h1>
  <table id="example" class="display" style="width:50%; float:left;"></table>

<script src="/static/pouchdb.min.js"></script>
  <script src="/static/app.js"></script>

</body>
</html>
```

Build Viewer with PouchDB (JavaScript)

```
const db = new PouchDB('csv_database');

// Funktion zum Löschen aller vorhandenen Daten in der PouchDB-Datenbank
function clearDatabase() {
  return db.allDocs({ include_docs: true })
    .then(result => {
      const docs = result.rows.map(row => ({
        _id: row.doc._id,
        _rev: row.doc._rev,
        _deleted: true // Markiere Dokumente als gelöscht
      }));
      return db.bulkDocs(docs); // Lösche alle Dokumente in einem Bulk-Vorgang
    });
}
```

Build Viewer with PouchDB (JavaScript)

```
// Funktion zum Abrufen von CSV-Daten aus PouchDB und Anzeige in der Tabelle
function displayCSVData() {
  db.allDocs({ include_docs: true })
    .then(result => {
      const csvData = result.rows.map(row => row.doc);
      processData(csvData); // Rufen Sie die Funktion processData auf, um die Daten anzuzeigen
    })
    .catch(error => {
      console.error('Fehler beim Abrufen der CSV-Daten:', error);
    });
}
```

```
// Funktion zum Importieren von CSV-Daten in PouchDB
function importCSVToPouchDB(csvData) {
  return clearDatabase() // Löschen Sie zuerst vorhandene Daten
    .then(() => {
      // Importieren Sie die CSV-Daten wie zuvor
      const lines = csvData.trim().split('\n');
      headers = lines[0].split(';').map(header => header.trim());
      const docs = lines.slice(1).map(line => {
        const values = parseCSVLine(line);
        const doc = {};
        headers.forEach((header, index) => {
          doc[header] = values[index];
        });
        return doc;
      });

      // Dokumente in PouchDB-Datenbank speichern
      return db.bulkDocs(docs);
    })
    .then(() => {
      console.log('CSV-Daten erfolgreich in PouchDB importiert.');
      displayCSVData(); // Nach dem Importieren die Daten verarbeiten und in der Tabelle anzeigen
    })
    .catch(error => {
      console.error('Fehler beim Importieren der CSV-Daten in PouchDB:', error);
    });
}
```

```
// Funktion zum Verarbeiten der Daten und Anzeigen in der Tabelle
function processData(dataSet) {
  const tableData = dataSet.map(data => {
    const rowData = {};
    Object.keys(data).forEach(key => {
      rowData[key] = data[key];
    });
    return rowData;
  });

  const columns = Object.keys(dataSet[0]).map(column => {
    return { title: column, data: column };
  });

  var table = $('#example').DataTable({
    data: tableData,
    columns: [
      { title: headers[0], data: columns[0].data },
      { title: headers[1], data: columns[1].data },
      { title: headers[2], data: columns[2].data },
      { title: headers[3], data: columns[3].data },
      { title: headers[4], data: columns[4].data },
```

```
76        { title: headers[3], data: columns[3].data },
77        { title: headers[4], data: columns[4].data },
78      ],
79      paging: true,
80      dom: 'Pfrtip',
81      initComplete: function () {
82        this.api().columns().every(function () {
83          var column = this;
84          var header = $(this.header());
85          var input = $('<input type="text" placeholder="Search" />')
86            .appendTo($(header))
87            .on('keyup change', function () {
88              if (column.search() !== this.value) {
89                column.search(this.value).draw();
90              }
91            });
92        });
93      },
94      searching: true,
95      mark: true
96    });
97  }
```

Build Viewer with PouchDB (JavaScript)

```
138  // CSV-Daten aus der Datei edak.csv laden und in PouchDB importieren
139  fetch('/static/test.csv') // Stellen Sie sicher, dass der Pfad zur CSV-Datei korrekt ist
140    .then(response => response.text())
141    .then(importCSVToPouchDB)
142    .catch(error => {
143      console.error('Fehler beim Laden der CSV-Datei:', error);
144    });
```

Build Viewer with PouchDB (JavaScript)

EDAK 2

Search:

Edition	Language	Region	Place	Inscription type
Anderson, JHS 19 (1899) 116, Nr. 109	Greek	Galatia	Kinna	funerary inscription
Anderson, JHS 19 (1899) 116, Nr. 110	Greek	Galatia	Kinna	funerary inscription
Bosch, Ankara 290	Greek	Galatia	Ancyra	building inscription
Bosch, Ankara 291	Greek	Galatia	Ancyra	building inscription
Bosch, Ankara 292	Greek	Galatia	Ancyra	building inscription
Bosch, Ankara 293	Greek	Galatia	Ancyra	building inscription
Bosch, Ankara 305	Latin	Galatia	Ancyra	honorary inscription
Bosch, Ankara 306	Greek	Galatia	Ancyra	building inscription
Bosch, Ankara 54	Latin	Galatia	Ancyra	building inscription
CIG 4045	Greek	Galatia	Ancyra	religious inscription

Showing 1 to 10 of 98 entries

Previous 1 2 3 4 5 … 10 Next

Why a Generic Viewer?

- Many projects share similar requirements
- Rewriting viewers is inefficient
- Generic components improve reuse
- Configuration enables customization
- Same system, different datasets
- Reduces development effort

→ **Reusable infrastructure for research projects!!!**

Generic Viewer (JS)

- A Generic Viewer is a reusable JavaScript-based data viewer
- Designed for structured research datasets (Input: CSV(s))
- Works with project-specific configurations
- Separates data, logic, and presentation
- Enables filtering, searching, and browsing

→A generic information system component

Generic Viewer (index.html)

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="icon" href="/favicon.ico" type="image/x-icon; charset=binary" />
        <title>Dev &ndash; GenView</title>
    </head>
    <body>
        <h1>Development Server &ndash; Generic CSMC Viewer</h1>
        <ul>
            <li><a href="./viewer/">Demo</a> (Toy data, but shows all features)</li>
            <li><a href="./docs/" target="_blank">Dokumentation (JS, Viewer) &nearr;</a></li>
            <li><a href="./docs/csmcgen/" target="_blank">Dokumentation (Python, CSMCGen CLI) &nearr;</a></li>
        </ul>
        <ul>
            <li><a href="https://git.chai.uni-hamburg.de/CSMC-Files/Generic-Viewer" target="_blank">Repository
        </ul>
    </body>
</html>
```

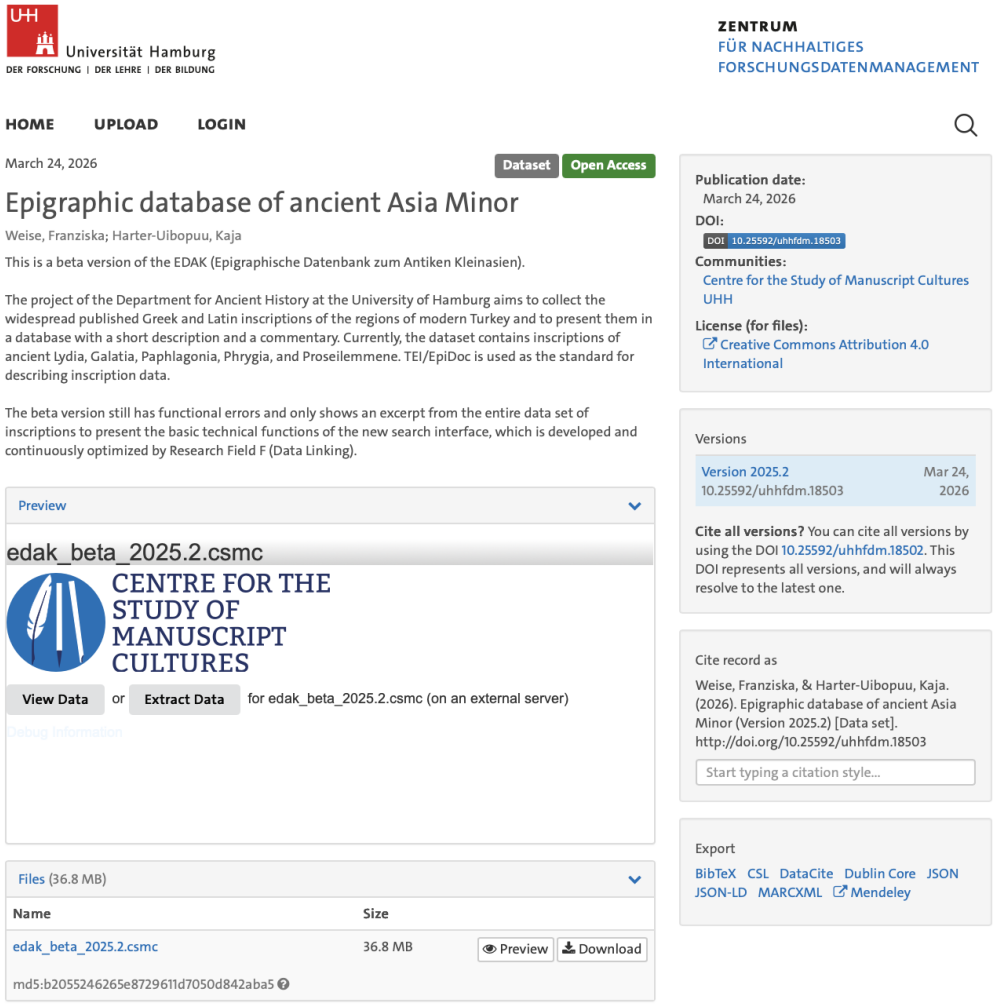
Generic Viewer

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="icon" href="/favicon.ico" type="image/x-icon; charset=binary" />
        <title>Dev &ndash; GenView</title>
    </head>
    <body>
        <h1>Development Server &ndash; Generic CSMC Viewer</h1>
        <ul>
            <li><a href="./viewer/">Demo</a> (Toy data, but shows all features)</li>
            <li><a href="./docs/" target="_blank">Dokumentation (JS, Viewer) &nearr;</a></li>
            <li><a href="./docs/csmcgen/" target="_blank">Dokumentation (Python, CSMCGen CLI) &nearr;</a></li>
        </ul>
        <ul>
            <li><a href="https://git.chai.uni-hamburg.de/CSMC-Files/Generic-Viewer" target="_blank">Repository
        </ul>
    </body>
</html>
```

Generic Viewer (viewer_config.js)

```
41  const VIEWER_CONFIG = {
42      // css (jQuery) selector of html element, the viewer populates in
43      "element" : "div#viewer",
44      // de-/active languages
45      "languages" : [
46          "en",
47          "de"
48      ],
49      // run the viewer in debug mode (uses less caching, etc.)
50      "debug" : true,
51      // the base url/ path (with trailing slash)
52      "base" : "./",
53      // location of assets file (relative to 'base')
54      "assets" : "static/assets.json",
55      // some information about the data shown by the viewer (will also be included in export)
56      "about" : {
57          // heading (applied to h1 of 'show=true') of the page
58          "heading" : "Demo of Generic CSMC Viewer",
59          // title of the page (applied if 'show=true')
60          "title" : "Demo CSMC Viewer",
61          // authors of the data
62          "authors" : "Magnus Bender",
```

Viewer in RDR



Architecture Evolution

Stage	Technology	Limitation
Static files	CSV/HTML	No interaction
DataTables	Frontend-only	Poor scalability
API + Viewer	JavaScript	Better interaction
PouchDB/CouchDB	Distributed DB	More complexity
Generic Viewer	Config-driven	Requires structured data

Conclusion

- Data transformation alone does not create an information system
- JavaScript enables interaction, querying, and visualization
- Repositories provide storage and interoperability
- Viewers connect users with research data
- Scalability requires architectural decisions
- PouchDB and CouchDB support distributed, offline-capable systems
- Generic viewers enable reusable research infrastructures

- **Research data becomes useful through interaction.**
- **Python prepares the data. JavaScript makes it usable.**