

Prosopographia Palmyrena: A Different Approach towards the Implementation of a Digital Prosopography

Jérôme Agater

Abstract: This essay outlines the software implementation part of *Prosopographia Palmyrena*, a digital prosopography database application. Operating with limited resources, we harnessed modern web technology to build an interface that enables users to explore Palmyrene inscriptions and their relationships. We describe the merits of using spreadsheets for data accumulation during our research phase and explore the complexities of database implementation. Our different approach centres around a server-less, database-less application integrating modern web technology, static site generation, and client-side interaction to yield a user-friendly platform for navigating the data corpus. Additionally, we describe some selected features that we were able to implement. As outlook the essay describes some features we want to implement in the future.

Introduction

There are well-regarded implementations of digital prosopography databases like *TMpeople*¹ and *DPRR*.² These projects have matured and grown over many years or are implemented by a dedicated department. The young *Prosopographia Palmyrena* has a much smaller team behind it, consisting only of one researcher gathering data from the vast corpus of Palmyrene inscriptions, and one developer who joined late in the project. With limited resources available, we chose a different technological approach to implement a subset of interesting features for researchers and other users. This paper describes some of the challenges for small research teams and presents our unconventional solution using modern web technology to deliver a database-less, server-less application as an interface for browsing and searching through the data corpus.

Challenges for Small Research Teams

Databases and their Database Software

Databases are a complex topic, whether in computer science or in a historical context. For small research teams working on a project to create a digital prosopography application, often no developer is available at all. In the market there are no common ready-made solutions to publish a prosopography to the web. Not many universities in Germany have established a department for the digital humanit-

1 <https://www.trismegistos.org/ref/> (consulted 29.04.2025).

2 <https://romanrepublic.ac.uk> (consulted 29.04.2025), excerpt from the introduction: “The *Digital Prosopography of the Roman Republic (DPRR)* is the result of a three-year, AHRC-funded project based at the Department of Classics and the Department of Digital Humanities at King’s College London.”

ies.³ Consequently technical expertise from data- or computer-scientists is often hard to come by and may not be available up front for many projects. Heuristically, it's useful to concentrate on the tasks the researcher needs to do anyhow. Therefore the work on data acquisition and gathering begins before technical challenges can be faced.

For many decades database software is dominated by relational database⁴ concepts⁵. The *DPRR* uses a relational approach for example, see fig. 1. While the current systems allow the storage and query of data, the upfront setup of such a database can be a challenge in itself. The prominent interface for these databases is the formal language SQL.⁶ This textual access method is cumbersome to use for people without prior experience in data science or who are used to a graphical user interface. To present the data in a queryable form, a graphical application is the way to go, allowing users to take full advantage of the assembled data.⁷ Obviously, another layer of software increases the complexity of the entire system. Furthermore once a project transitions from the collection and research phase to operational use, the list of tasks to keep the database and application software running only grows: maintaining the underlying subsystems, the frameworks used in the application, the application code and servers is required to ensure the continuous operation of the service.

For *Prosopographia Palmyrena* the challenges we faced were accordingly twofold:

1. The database application needs to run without constant need for maintenance for many years.
2. The database application should present a user-friendly interface to query the underlying digital prosopography data via the web.

A Step Forward with Spreadsheets

If a database consists only of uniform records tracked and stored in one table, a spreadsheet software can be used perfectly fine as an editor for the data in its tabular form⁸ during the research phase (see fig. 2 for an illustration of the record structure in *Prosopographia Palmyrena*).

However, spreadsheets have some drawbacks:

- In collaborative settings, update anomalies are possible when researchers attempt to update the same records, resulting in one update silently overwriting the change-set of the other. Additionally, conflicts can arise when the data is updated in parallel, preventing one change from being committed to the database and requiring the author whose transaction failed to manually resolve any conflicts. This is assuming it is even possible to open two concurrent editing sessions for the database table at all.

3 I.e., an established department like the Department of Digital Humanities, Faculty of Arts & Humanities, King's College London, <https://www.kcl.ac.uk/ddh> (consulted 29.04.2025) in UK.

4 Relational database software works as an engine to create, fill, update and manage tables of tuples containing data atoms like numbers or strings of characters. The tables of a database then connect or relate to one another via *joining*; the values for an identifier of one table map to the values of an identifier of another table, creating a relation between the two tables. For arbitrary N:M connections, a third table can be used connect two tables by mapping the identifier of one table row to an identifier of the other table's row in the same way.

5 Like MySQL and PostgreSQL servers in the open-source world, FileMaker and Microsoft Access in home and office settings, or commercial databases like IBM's DB2 or Oracle in business settings.

6 Structured Query Language.

7 Other methods do exist though, for example *DPRR* presents an RDF-based interface.

8 Usually Rem.: then in an unnormalized form.

- Spreadsheets can also be extremely slow when handling large datasets.
- Office spreadsheets like Microsoft Excel and LibreOffice/OpenOffice Calc have limits on the maximum length of tables and number of records they can handle.
- Working with many interrelated tables is not well supported, if supported at all.
- While learning SQL might be complex, it also enables the execution of complex ad-hoc data queries in a standardized way. As a result, many software frameworks provide common support for various database solutions using SQL – which is not the case for spreadsheets.

However, storing the data in a table is generally a very good way to accumulate the data in the first place, and spreadsheets have some advantages, too:

- They are a known and generally well understood tool. Researchers are accustomed to using spreadsheets in various settings and situations.
- Additionally if the dataset is below 100,000 rows, considerations such as size, speed, and data types are usually irrelevant because today's machines are just fast enough. The size of the problem domain fits well within the comfortable range for these systems.
- If only a few researchers are updating the data set,⁹ and updates are generally not performed concurrently, conflicts and anomalies are reasonably unlikely and/or can be quickly addressed. Considerations that are valid for 'high volume – fast paced' setups are not applicable in this situation.
- Another advantage is the absence of problems with the operation of the system:
 - A spreadsheet application is a self-contained system that can be maintained and set up quickly by a single researcher.
 - The system works offline, eliminating the need for an internet connection.
 - Additionally, by copying a single file, backups are both quick and easy, while divergent branches of the dataset can be derived in the same way for experimentation.
 - Sharing a snapshot of the entire database can be achieved by providing the file via email or otherwise.

Appropriately, a spreadsheet has been successfully used to store the record corpus of *Prosopographia Palmyrena* during the primary research and gathering phase.

⁹ For our *Prosopographia Palmyrena* Peter von Danckelman did all the work.

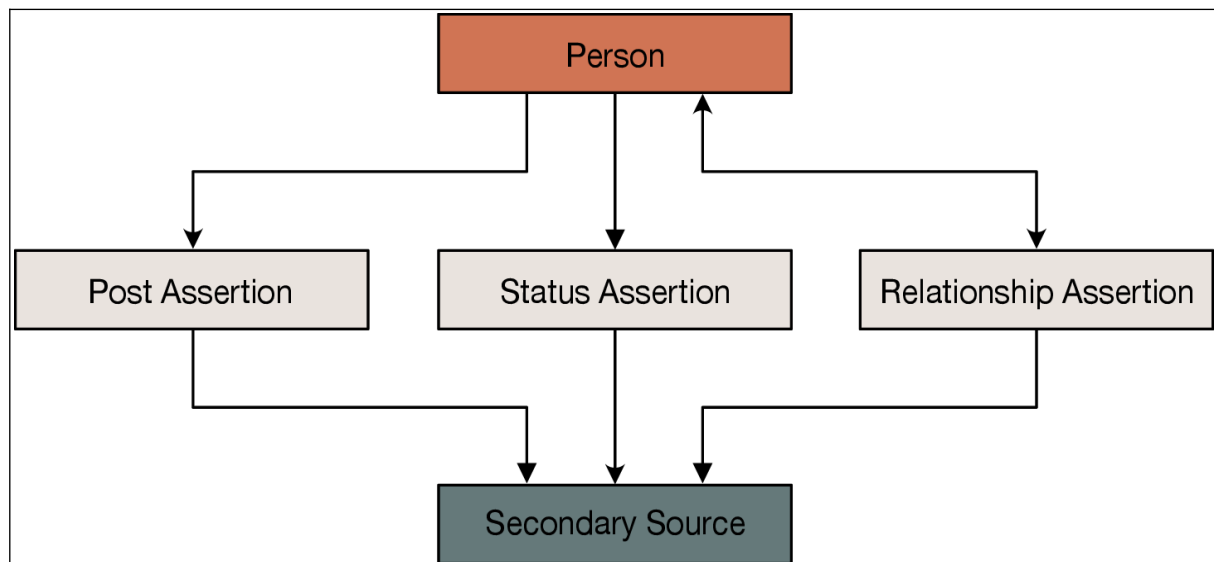


Fig. 1: Table and relational structure of *DPRR* (redrawn from FV [2017]).

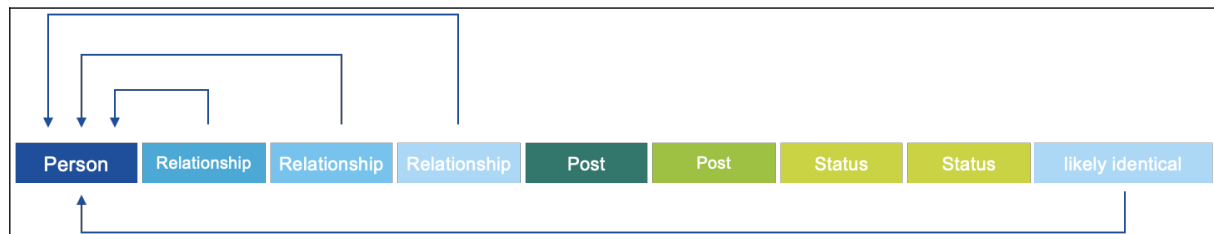


Fig 2: Single self referential table structure of *Prosopographia Palmyrena*.

Making a Dataset Accessible on the Internet

Once a dataset with the corresponding table(s) has been created, how can it be published so that researchers can benefit from the available corpus?

For a spreadsheet-contained database, there are some freely available ways of publishing a version on the web:

1. The file containing the spreadsheet data could be shared in its raw form through platforms like Google Sheets or Microsoft Office 365, enabling others to access and/or edit the table. The spreadsheet implementations from Google or Microsoft would then be utilized to query the dataset. The file could also be downloaded from these platforms in raw form and opened in a local spreadsheet software, again such as Microsoft Excel or OpenOffice/LibreOffice Calc.
2. The dataset could be migrated to an existing but more sophisticated and demanding solution like the application that runs the *DPRR*. It requires its own server setup to run a Python application and a relational database.
3. Alternatively, a new solution could be developed for the dataset, and then the data could be migrated or imported into the newly implemented system. However, realistically, this approach is often too challenging for most teams that lack developers or have limited experience in application and database development and programming.

These apparent solutions come with their own challenges:

1. The database should be accessible to everyone. While spreadsheets offer a user-friendly interface for tabular data, they lack a proper query interface that accurately represents the data domain and its internal relationships.
2. Application servers are unavailable, i.e., running a *Ruby on Rails*¹⁰ or *Django*¹¹ application with an own database server on the infrastructure available with correct firewall settings, logging, etc. is friction and currently not really supported without hiring a dedicated system administrator or operator.
3. We don't have support for continuous operation, so as before, the system will have to run without intervention or constant observation.

Some additional characteristics of our project provide room for creating solutions:

- Given that the collection phase is mostly complete, we anticipate infrequent updates to the dataset. Consequently, there's no necessity for us to develop or maintain editing interfaces, as this would be a waste of resources given their expected usage.
- The dataset that has been compiled and researched from the text corpus is not extensive in the context of modern computing. Currently, *Prosopographia Palmyrena* consists of fewer than 10,000 records.

¹⁰ *Ruby on Rails* is a prominent framework; it is the technology used to implement AirBnB, GitHub, Hulu, and Shopify.

¹¹ *Django* is another prominent framework; it is the technology used to implement Disqus, DropBox, Instagram, and Spotify.

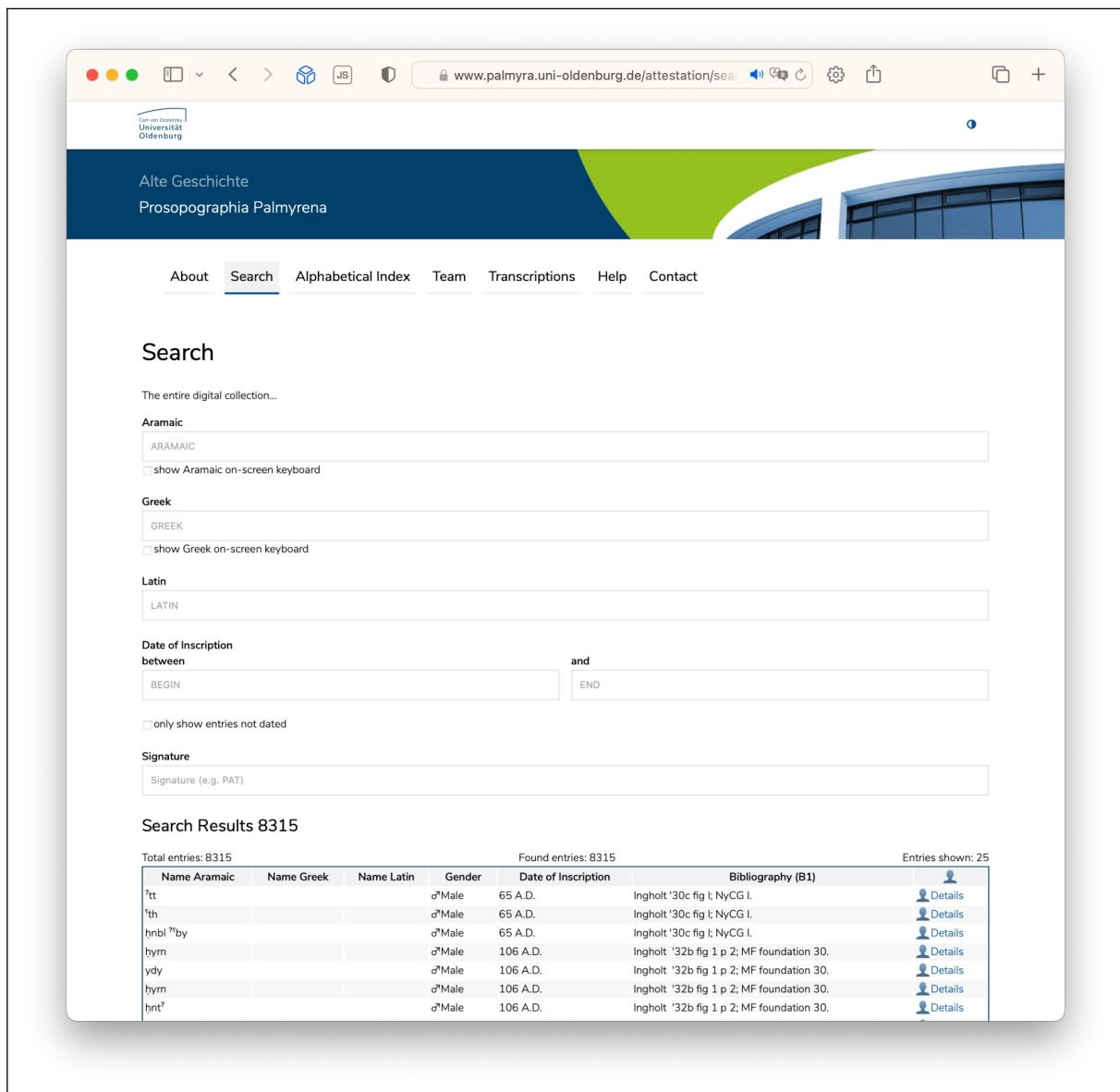


Fig 3: Primary search interface of *Prosopographia Palmyrena*.

A Solution Using a Different Approach

In the last decade, an old trend has resurfaced: *static site generators*. Instead of having an application run on the server and dynamically generate responses on the fly based on user queries and navigation, static pages are generated in advance from a data corpus. This corpus could be an existing catalogue or a content management system containing articles. Each entry in the corpus is then transformed into an HTML file within a path structure, already pre-rendered and populated with its data content.

These static files, representing the data corpus, are served through a conventional web server. This technique eliminates the need for an application to reside on the server, which means there's no requirement to maintain a database, periodically update an application framework for security, or constantly monitor servers for potential failure.

As long as a standardized and cost-effective static web server is available, the data can be effectively served. Furthermore, since high-performance static web servers are not resource-intensive, the likelihood of our servers being overwhelmed by Google, AI crawlers/spiders, other bots or over-eager users is greatly minimized.

A Blast from the Past

Interestingly, a similar approach has been employed by John Bradley et al. to publish the dataset of *Prosopography of the Byzantine Empire*¹² on a single CD-ROM. They achieved this by generating tens of thousands of HTML files. These HTML files encapsulate the entire data in a deconstructed form. Alongside this, a limited¹³ dynamic query system operating within the browser is initiated on every page. While the primary interactivity stems from navigating between the pages, this dynamic system facilitates the formulation of more intricate queries and introduces a degree of additional dynamism. Importantly, all these operations occur within the client's browser and therefore necessitate no specialized server or application.

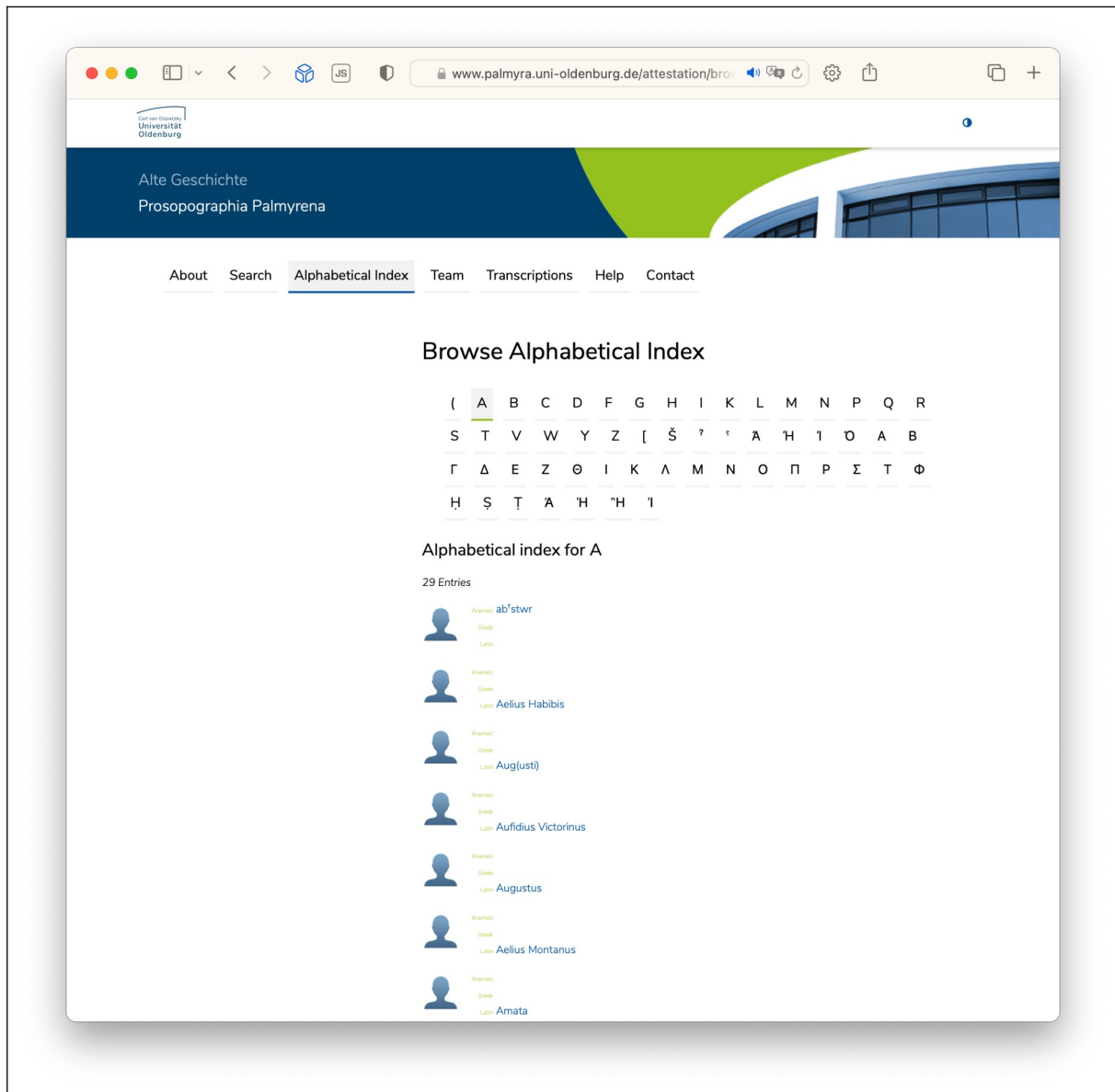


Fig 4: Index interface of *Prosopographia Palmyrena*.

We can utilize our spreadsheet to generate the entire site using the same methodology, thus satisfying the requirements for the first challenge concerning database software.

¹² <https://pbe.kcl.ac.uk/data/help/about.htm#intro> (consulted 29.04.2025).

¹³ Due to the constraints of that language's implementation at that time.

Realizing a Query Application

Considering the enhanced capabilities of today's browsers and client computers, we can move the application layer from the server into the client application as well. By doing so, we satisfy the requirements for the first challenge related to application software as well. Similar to the approach employed for the *Prosopography of the Byzantine Empire*, once a page is loaded, we can initiate dynamic client software that includes the necessary query engine. This way, we can offer comprehensive search options without the need for an application to be operational on the server (see figs. 3 and 4 for the user interface created). We envisioned that this approach would yield additional advantages, such as increased speed (since information requests no longer need to traverse between the client and the server) and enhanced interactivity. Indeed, this expectation has proven accurate.¹⁴

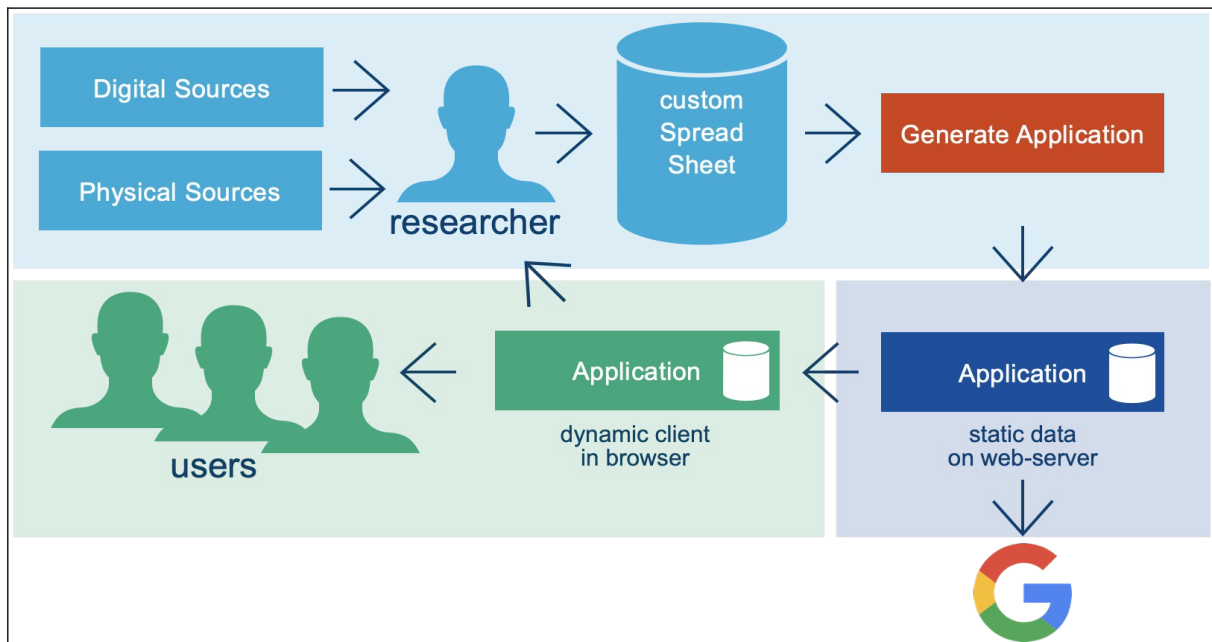


Fig. 5: Data- and workflow of our approach.

About the Enabling Technology

Following the mentioned resurgence of static site generators, the *NEXT.JS* framework (referred to as *Next.js* in subsequent paragraphs) emerged prominently. *Next.js*¹⁵ not only serves as a static site generator, but it also functions as a fully-integrated framework for our browser-based application.

Adopting an opinionated approach, *Next.js* generates the complete static setup, including a server-side section encompassing all HTML files, the associated path and folder structure, and optimizing assets such as images and style definitions in the form of CSS.¹⁶ Additionally, it can be employed to produce other HTML-like text-derived file formats, for example RDF documents¹⁷ for automated access and *Scalable Vector Graphics (SVG)* for generating figures or drawings.

¹⁴ As we can immediately react to events in the user interaction. For example, providing an on screen input for entering data or an interactive map for exploration of the geographical context of an attestation/person record.

¹⁵ *Next.js* is written in a typed superset of JavaScript called TypeScript. TypeScript is the de-facto standard of developing applications with the JavaScript language. TypeScript gets transformed to JavaScript via a transpilation step, where JavaScript acts as a transpiler target.

¹⁶ *Cascaded Style Sheets* the language for specifying the look (and partially behaviour) of elements on web pages.

¹⁷ Like they are available for *DPRR*.

By pre-rendering entry points for all dataset entries, *Next.js* provides a foundation for search engines, including Google, to crawl and index the corpus. Consequently, this significantly enhances the data's discoverability.

For *Prosopographia Palmyrena*, *Next.js*'s capacity to serve as the framework implementing dynamic client-side interaction has been particularly advantageous. It serves as the foundation on which we build the logic of our user-facing application and querying interface.¹⁸

The sequence of actions unfolds as follows: Whenever a static page from the server is requested, the entire application is downloaded with it and automatically initiated by the browser. This brings the application back to life from a 'freeze-dried' state. This flow is illustrated in fig. 5.

Providing the Database

The rows of the database table, originally derived from the spreadsheet, are extracted and then compressed and consolidated into a single structured text file for storage, specifically in the widely used JSON¹⁹ format.

The complete dataset is downloaded into the client application as immediate, fixed data.²⁰ However, this approach is not scalable to hundreds of thousands of rows due to size and memory constraints. Nevertheless, the JSON storage file is approximately 2.7MB in size, which is still smaller than a typical mobile photo or video advertisement.

After the data has arrived in the client, the JSON data is initially deserialized into plain JavaScript objects, which are subsequently reconstructed into modelled entities. These modelled entities are themselves plain JavaScript objects and serve a purpose similar to the *Django* models, *RoR ActiveRecord*s, or *Java Hibernate* instances, that typically constitute the data layer for a server-side application. Every entity, such as an attestation, possesses a unique ID corresponding to the original row number in the source spread-sheet. This ID functions as primary key.²¹ Attestations can cross-reference each other using their unique IDs. From the Palmyra corpus, a few hundred individual persons were identified by Peter von Danckelman; each person's entry has references to their associated attestations using these IDs.

Updating the Database

Rather than offering a distinct editing interface for this system, the spreadsheet remains the primary source for the *Prosopographia Palmyrena* database. Whenever changes occur within this spreadsheet due to data updates, the dataset within it is leveraged to regenerate the static HTML files (amounting to less than 10,000) in a matter of seconds. Subsequently, we can deploy the static dataset to our static web server, rendering it instantly accessible to users and enabling Google and other search engines to index it.

18 Using the principle *inversion of control*, where the framework provides the flow structure while the users define the actions to take within said flow.

19 *JavaScript Object Notation*, a textual file format to store structured data consisting of numbers, strings, dictionaries and lists often used in the web context. An example looks like this: {"author": "J  r  me", "publication": "Digital Classics Online", "date": 20230801}.

20 It's also thinkable to store everything in an SQLite database, push that to a server and use one of the HTTP-Range queries to fetch specific rows. This approach is described in a blog post from 2021 here: <https://phiresky.github.io/blog/2021/hosting-sqlite-databases-on-github-pages/> (consulted 29.04.2025).

21 An approach also originally used by *Ruby on Rails*' well-regarded *ActiveRecord* data-modelling subsystem.

Data Querying the Base

In our solution the querying is achieved using the native methods of the programming language (JavaScript) rather than a dedicated framework or query language.

Over the past decade, the methods for querying object graphs (or rather: arrays of objects) in JavaScript have evolved significantly. While the available methods might not be as comfortable for the programmer as other interfaces,²² they demonstrate reasonable speed (given our data-size) and straightforwardness. In modern browsers, object queries – even through large datasets – are optimized, requiring only a few fractions of a second to search through many thousands of rows. The more intricate operations found in professional database systems don't particularly offer an advantage in this context, given that the results are presented instantaneously anyway.

However the reification of queries is not inherently provided, meaning an abstract representation of a query isn't readily available. Because of this, we needed to construct our own notion of a query. Consequently, it takes more time to make a query formulate able for our application's interface.

What might typically constitute a simple query, possible relational in nature, becomes another generic search in our current solution. For instance, when a person possesses a *childname_in_aramaic* attribute (see fig. 6 for context of use), locating all potential children of that attestation's person involves executing a search for that name and subsequently displaying the results. It's then up to the user, drawing from their domain expertise, to determine whether the attestations represented in the results correspond accurately.

In the future, a more refined approach could involve referencing the child through its specific unique ID, as discussed earlier – moving back towards the relational database approach.

22 Like SQL, Microsoft's *Linq*, the methods of *ActiveRecord* from *Ruby on Rails*, or the *Django* query manager.



Fig. 6: Relationship diagram for every attestation.

Visualizing Locations

A considerable number of entries within the *Prosopographia Palmyrena* offer some form of geographical context. This context might be derived from place names mentioned in attestations, the inscriptions' physical locations, or external sources.

- Certain entries possess geographic information directly associated with them.
- For some, only contextual locations data is available.
- However, certain entries lack any form of ascertainable location.

Whenever a direct geographic reference exists, it can simply be displayed. In situations where explicit information is absent but contextual details are available, the context can be presented in place of concrete geographic information. The dots in fig. 7 mark all locations present within the application's data.

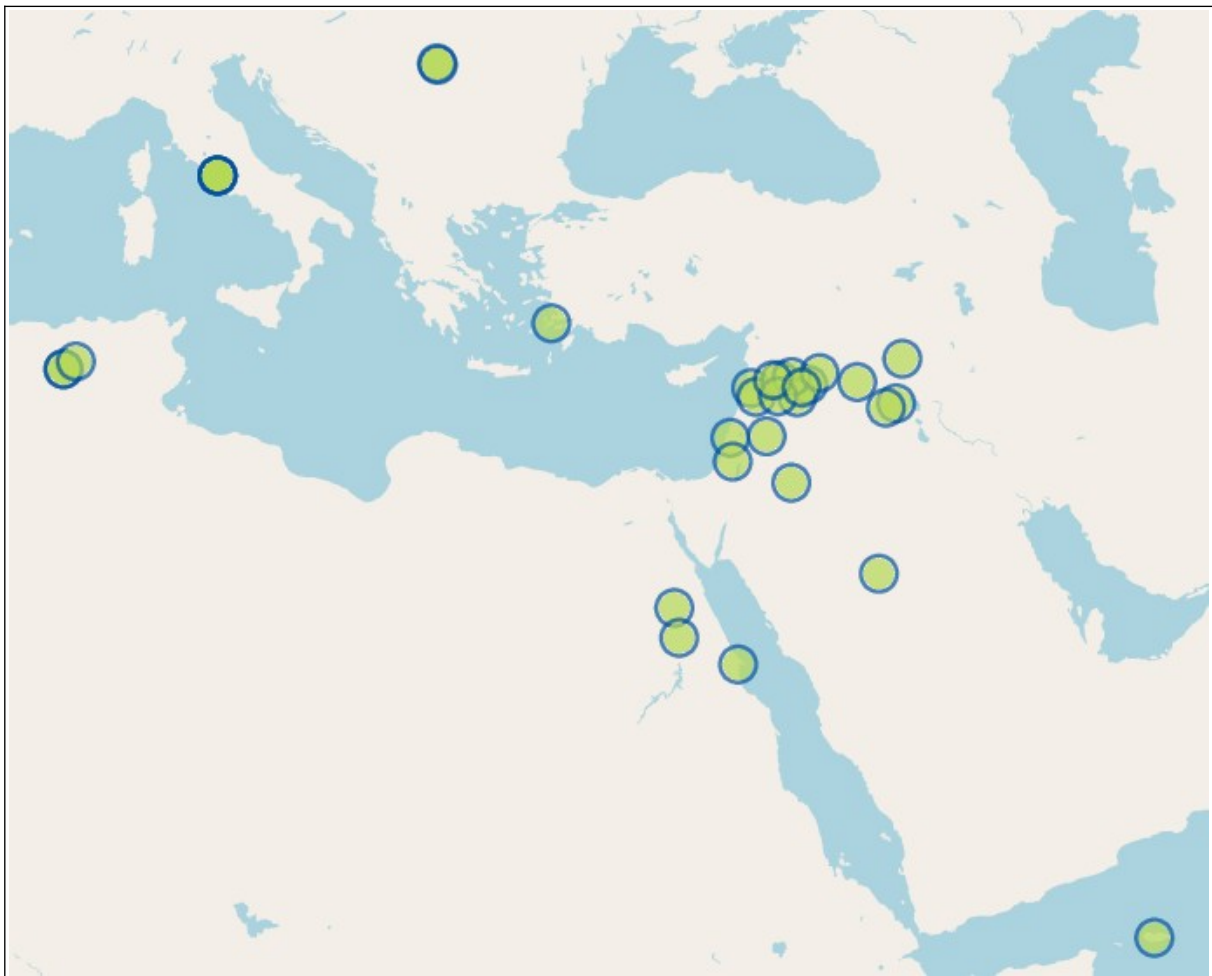


Fig. 7: All approximate locations in *Prosopographia Palmyrena* with base map from OSM.

GPS Coordinates

The locations are internally mapped to GPS coordinates for display on an interactive map. We employ the widely used WGS84 world geographic system as the foundation for these coordinates. This system is renowned for its utilization in GPS and platforms like Google Maps.²³

²³ In addition to the conventional representation using degrees, minutes and seconds (e.g. 53°9'N, 8°13'O), the use of decimal-degree is becoming more prevalent (e.g. 53.14684981205692, 8.184430032093898). This decimal-degree notation can be stored with high efficiency, within acceptable inaccuracies inherent to floating point numbers.

Map Data

We employ the collected data along with map images generated from the *OpenStreetMap Project* (OSM) for our base map (also used in fig. 7). OSM operates as a collaborative mapping initiative where volunteers contribute, update and expand mapping information, similar to Wikipedia. The mapping data is governed by the *Open Database License* (ODbL), permitting access, usage, and contributions.²⁴

To visualize the georeferences, we utilize the *OpenLayers* library.²⁵ This well-established library was created and is used by OSM itself. *OpenLayers* offers the necessary software infrastructure for map interaction, including functions like zooming, panning, and handling the retrieval of base map data. It also provides the API to present geographic features such as markers for our locations. Being open-source software, *OpenLayers* can be extended and modified as needed.

Shareability

Our objective is to ensure that every entry, location, and search within our system can be referenced by a URL,²⁶ allowing researchers and others to directly access linked information. In a way, this is our first step towards a constellation of prosopographies in a linked data infrastructure.

When a link is shared through messaging apps, social media posts, or other mediums, the tool responsible for presenting the shared content can generate a comprehensive preview of linked URL. Many common messaging clients, including Apple Mail, WhatsApp, Facebook, and X (Twitter), do this.

The preview might encompass plain text, an image or even a video, often displayed in the form of a card-like visual. In *Prosopographia Palmyrena*, the ‘card’ symbolizing the shared entity already contains some relevant information. This concept serves as a homage to the index cards utilized in past filling cabinets. To achieve this, we render an image integrating relevant textual details along with the project’s name and originating institution.²⁷

This approach ensures that even our concise URLs offer immediate context. Consider fig. 8, which portrays a fictional conversation between two researchers discussing an attestation regarding the children of Νεσῶ using Apple’s *iMessage* application. The URL²⁸ is automatically retrieved by *iMessage*, the location of the generated image is extracted, and the generated image is showcased as a card, supplying the aforementioned context information.

24 <https://wiki.osmfoundation.org/wiki/Licence> (consulted 29.04.2025).

25 <https://openlayers.org> (consulted 29.04.2025).

26 For example <http://www.palmyra.uni-oldenburg.de/attestation/597> references an attestation for ῥῶδω (Soadu).

27 Abteilung Alte Geschichte, Department für Geschichte, Carl von Ossietzky Universität Oldenburg.

28 <http://www.palmyra.uni-oldenburg.de/attestation/729> (consulted 29.04.2025).

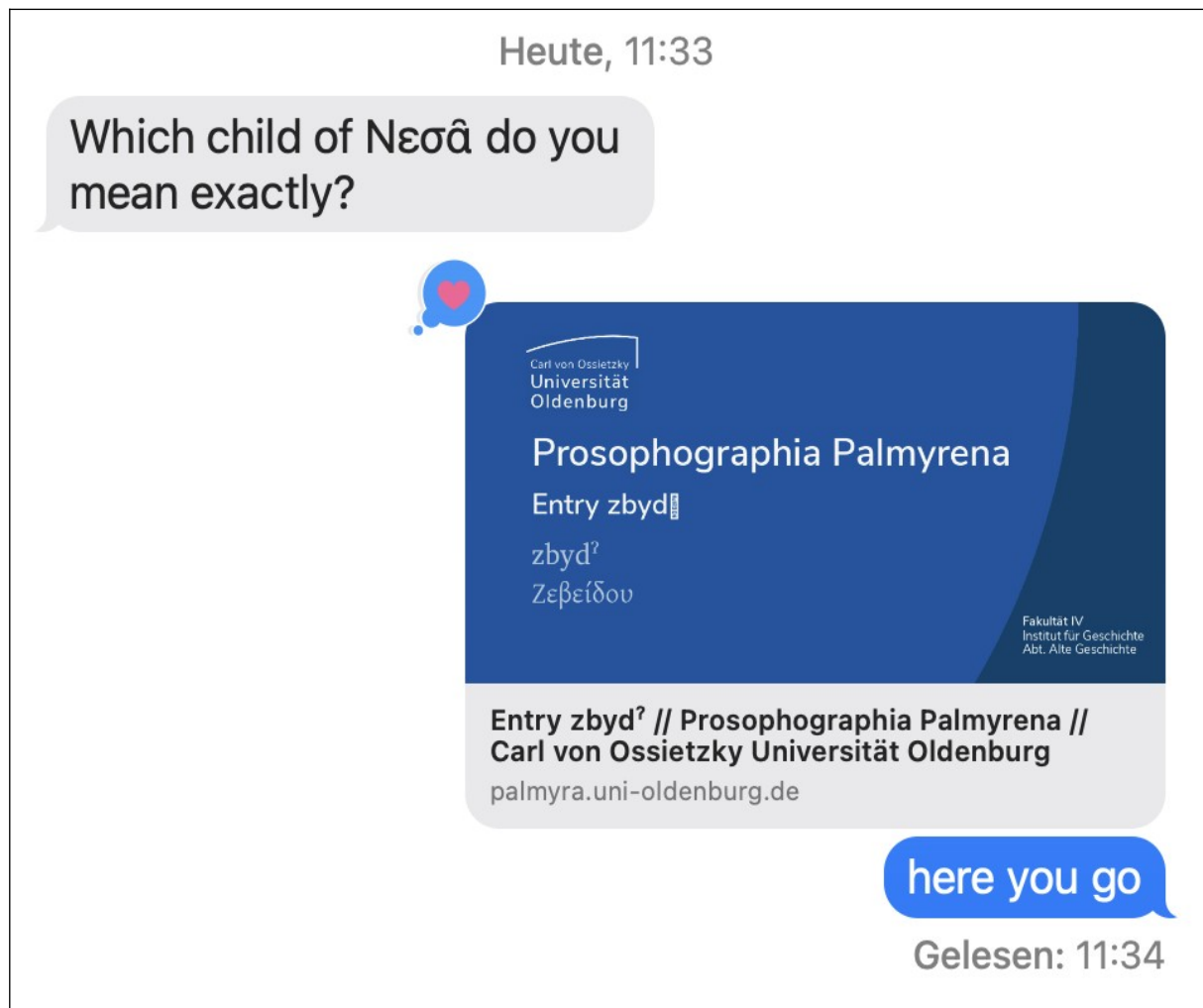


Fig. 8: A possible dialogue between two researchers using Apple's *iMessage*.


Realizing Aramaic Input

Text is comprised of characters,²⁹ yet computers store text as sequences of numbers. These numbers, in turn, represent something else than their value – like the characters. Each character can be assigned a specific number, enabling the computer to reference it. However, there are different schemas to agree on what character is represented by which number.

Historically, when computer resources were more limited, a single number represented a single character in the West. Traditionally, the byte ranging from 0 to 255 formed the foundational unit of the number system. Character encodings were then devised to fit all characters within these 255 slots. The renowned ASCII text encoding for instance, covers the Latin alphabet and certain control codes. However, the world utilizes a much larger array of characters than ASCII can hope to accommodate. European scripts, which include only some special characters, often fit within the 255 possible values of a byte. Yet, Asian scripts like Japanese and Chinese demand a much greater number of 'characters'. As a result, encodings were designed that use more than one byte to represent a single character.

29 The relationship between text, characters, and individual glyphs is of course more complex, but for our understanding the abstraction level of assuming text to be assembled from characters is sufficient.

Older languages and scripts like Aramaic weren't initially considered by implementers.³⁰ Consequently, these uncommon scripts generally lacked defined encodings and couldn't be effortlessly represented on computers. Researchers addressing these challenges have employed various strategies:

- Some researches have developed ad-hoc encodings tailored for their specific needs.
- Others have utilized fonts, that map a Latin character (e.g. 'A') to an entirely distinct glyph like ³¹ which they actually intended to display.

Font Approach

A notable instance is the collection developed by Ulrich Seeger³²: He designed numerous fonts³³ for the transcription of Semitic texts. With the font approach all individuals working on a transcribed text must use the same encoding and have the specific font installed in order to accurately read the text with its intended characters. Within a confined working group, achieving this consistency can be managed by configuring all computers of the group accordingly. However, when the audience extends to the global internet, accommodating an array of diverse and often constrained systems becomes unrealistic.

Solution: Today's *Unicode*

The *Unicode* system assigns a unique number to each character/glyph. In *Unicode Version 15*, 149,186 characters are defined.³⁴ Naturally, this exceeds the capacity of representation by a single byte. To address this, encodings for *Unicode* employ multiple numbers to represent a single character.

Unicode characters are entered through specialized keyboard layouts (e.g. English, German, Japanese etc.) or potentially by inputting their individual numeric values. The latter approach can be slow and cumbersome, as it necessitates the input and recall of lengthy numbers, just to represent a single character.

In principle, users should be able to compose search queries using characters from the script/language, with the computer competently handling the text. However, in practice, users often encounter difficulties in typing specific characters and may need to resort to entering *Unicode* numbers instead. Custom keyboard layouts can be devised to mitigate this issue; for instance, Ulrich Seeger created a keyboard layout for macOS enabling direct character input (see fig. 9).

Users frequently encounter obstacles when their systems are restricted. In settings like public libraries, users lack control over available input methods and are unable install software.

30 Indeed, numerous companies within the business realm focused primarily on rendering their predominantly English/European texts feasible for representation.

31 Hieratic sign A1 from Papyrus Petersburg 1115, taken from <https://commons.wikimedia.org/wiki/File:Hieratic-sign-A1.svg> (consulted 29.04.2025).

32 Ulrich Seeger, Universität Heidelberg, Semitistik, Seminar für Sprachen und Kulturen des Vorderen Orients <https://www.uni-heidelberg.de/fakultaeten/philosophie/ori/semitistik/seeger.html> (consulted 29.04.2025).

33 You can find the fonts and a description from Ulrich Seeger on his website: https://www.uni-heidelberg.de/fakultaeten/philosophie/ori/semitistik/seeger_fonts.html (consulted 29.04.2025).

34 <https://unicode.org/versions/Unicode15.0.0/> (consulted 15.09.2023).

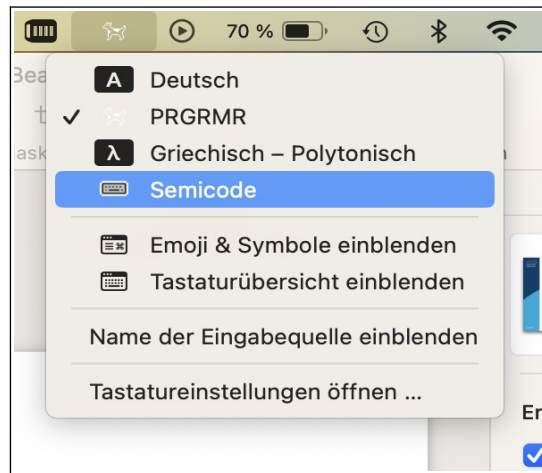


Fig. 9: *Semicode* keyboard layout for macOS-based Systems to insert Aramaic *Unicode*.

Our Solution: Provide an On-Screen Keyboard

We opted for an on-screen keyboard to allow users to input Aramaic and Greek script, without having to activate or install extensions (or change any settings at all). Users can either directly type the characters, if their system allows this – or they can use the mouse to enter special characters from the text systems (see fig. 10 for the entire keyboard layout for Greek and Aramaic).

Searching through *Unicode* Texts

Within *Unicode*, certain characters can be formed from different combinations of glyphs. For a single character, there might be multiple ways of forming the corresponding visual shape with *Unicode*. One simple example is the German umlaut ‘ü’. There is a dedicated glyph for this character, but the same shape can also be constructed by combining the glyph for ‘u’ with the diacritical glyph ‘̈’.

When users search through the database, they typically input a character in one specific way or another. Consequently, our application standardizes each text into a canonical form during the search process. This way, the original form of the character used for the description becomes irrelevant. To achieve this, a standardized *Unicode* normalization algorithm is employed, available as efficient native code through the web browsers.

During the workshop *Prosopography in the Digital Age* Yanne Broux³⁵ highlighted an additional challenge concerning Greek text. Due to the usage of polytonic Greek, multiple variations of a letter, such as ‘α’, exist. This variance arises from users and researches inputting Greek according to their preferred spelling. As users might inconsistently use or omit diacritics, addressing this issue is a consideration for the future.

35 KU Leuven.

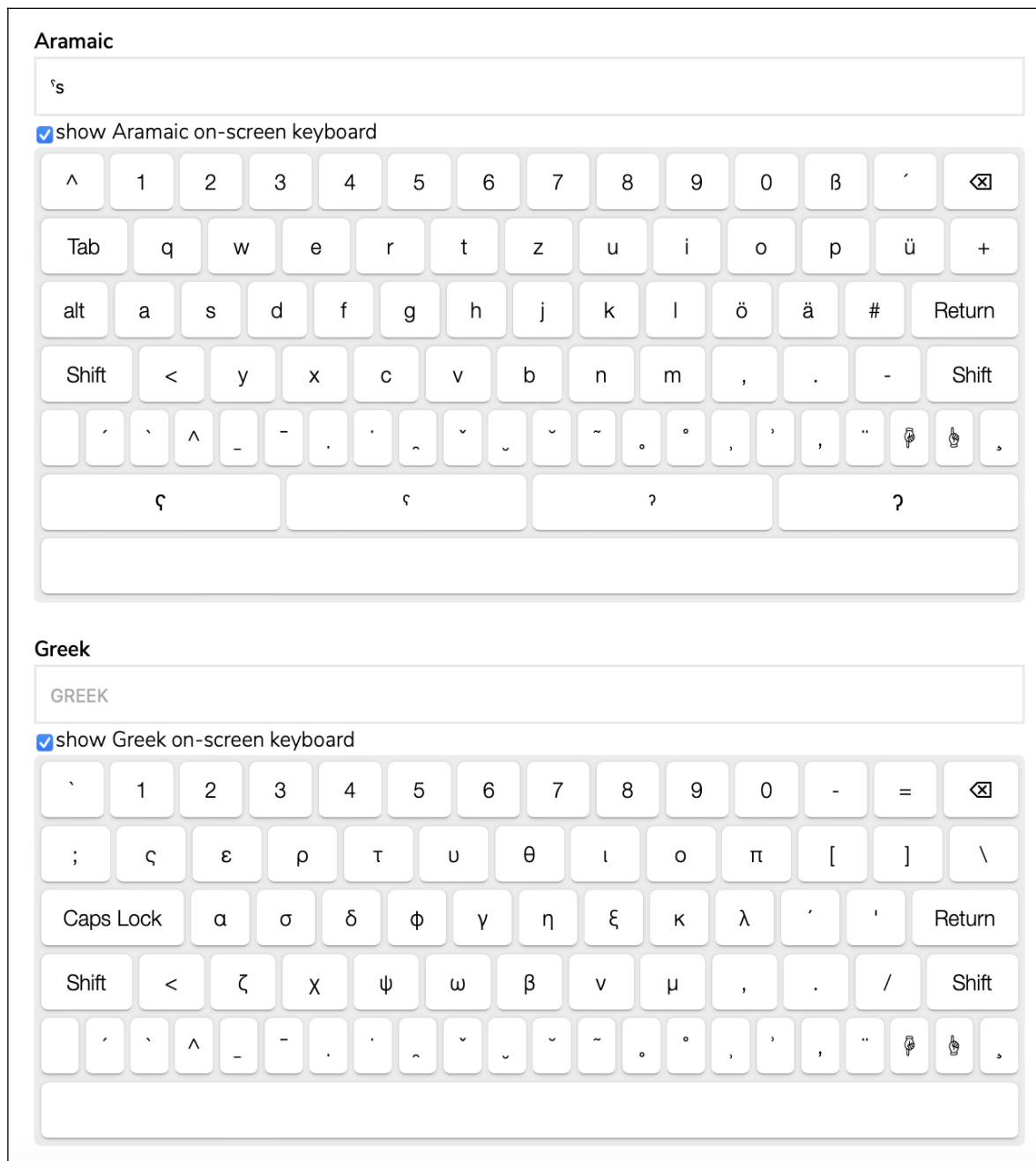


Fig. 10: On-screen keyboards for Aramaic and Greek in *Prosopographia Palmyrena*.

Outlook and Further Work

While the database can already be searched for the most relevant aspects, the set of attributes that can be queried is currently limited to the most prominent and relevant attributes. Ideally we want to extend the queryable set to match the entire set of attributes available. Furthermore the map representing all locations within the data could be used for filtering records via their location. This would allow users to visually explore the data in its geographical context. While the data stored inside the application can be searched and browsed, exporting the data set or subsets of the data to a text-, spreadsheet-, or other data file could be useful for researchers. Finally, it will be interesting to hear more from the users of the database application about their use-cases and their preferences, once the system is in regular use. The further development of *Prosopographia Palmyrena* will then be guided by the needs of users and researchers.

Abbreviations

SQL	Structured Query Language
RoR	Ruby on Rails
OSM	Open Street Map
OL	Open Layers
HTML	Hypertext Markup Language
DPRR	Digital Prosopography of the Roman Republic
JSON	JavaScript Object Notation

Sources

Online Sources

Figueira / FV (2017): L. M. Vieira, Modelling a Prosopography for the Roman Republic, The Digital Prosopography of the Roman Republic Project, online 2017, <https://dh2017.adho.org/abstracts/091/091.pdf> (consulted 29.04.2025).

<https://www.trismegistos.org/ref/> (consulted 29.04.2025).

<https://romanrepublic.ac.uk> (consulted 29.04.2025).

<https://www.kcl.ac.uk/ddh> (consulted 29.04.2025).

<https://pbe.kcl.ac.uk/data/help/about.htm#intro> (consulted 29.04.2025).

<https://phiresky.github.io/blog/2021/hosting-sqlite-databases-on-github-pages/> (consulted 29.04.2025).

<https://wiki.osmfoundation.org/wiki/Licence> (consulted 29.04.2025).

<https://openlayers.org> (consulted 29.04.2025).

<https://commons.wikimedia.org/wiki/File:Hieratic-sign-A1.svg> (consulted 29.04.2025).

<https://www.uni-heidelberg.de/fakultaeten/philosophie/ori/semitistik/seeger.html> (consulted 29.04.2025).

https://www.uni-heidelberg.de/fakultaeten/philosophie/ori/semitistik/seeger_fonts.html (consulted 29.04.2025).

<https://unicode.org/versions/Unicode15.0.0/> (consulted 29.04.2025).

Figure References

Fig. 1: Table and relational structure of *DPRR* (FV [2017]).

Fig. 2: Single self referential table structure of *Prosopographia Palmyrena*.

Fig. 3: Primary search interface of *Prosopographia Palmyrena*.

Fig. 4: Index interface of *Prosopographia Palmyrena*.

Fig. 5: Data- and workflow of our approach.

Fig. 6: Relationship diagram for every attestation.

Fig. 7: All approximate locations in *Prosopographia Palmyrena* with base map from *OSM*.

Fig. 8: A possible dialogue between two researchers using Apple's *iMessage*.

Fig. 9: *Semicode* keyboard layout for macOS-based Systems to insert Aramaic *Unicode*.

Fig. 10: On-screen keyboards for Aramaic and Greek in *Prosopographia Palmyrena*.

Author Contact Information³⁶

Jérôme Agater, M.Sc.

Carl von Ossietzky Universität Oldenburg

E-mail: j.agater@uni-oldenburg.de

³⁶ The rights pertaining to content, text, graphics, and images, unless otherwise noted, are reserved by the author. This contribution is licensed under CC BY-SA 4.0.