[WORKING PAPER] A prosopographical software package

Tom Bellens[1,⋆]

[1] *KU Leuven Public Governance Institute*

Abstract. In this working paper, we present an automated software package. This means that two pieces of software are presented: on the one hand, a software package for prosopographic projects and, on the other hand, the software that generates these packages. The software package will consist of a PostgreSQL database, a Spring Boot back-end and an Angular front-end and aims to simplify data collection within research teams and make it more precise. To make as few assumptions as possible about the shape of the data, each software package is generated based on a JSON in which the target data is modelled. This working paper presented at Politicologenetmaal 2023 in Leuven is a first attempt and serves mainly to gather feedback on the idea.

DISCLAIMER

This is a working paper. Please do not cite.

1 Introduction

In this paper, we present a semi-automated software tool for prosopographical research. The tool is semi-automated in the sense that the underlying source code is re-generated for each prosopographial project based on the user's needs. Before discussing how the generation of the source code works, we will very briefly go over what exactly prosopography is and why we think a software tool could be useful in the first place.

2 Prosopography and earlier software packages

Prosopography is an archaic-sounding term for a very simple idea: it is the study of the biographical backgrounds of a well-defined group, usually an elite. The method has its origins in 19th-century German historiography and focused on the political elites of different historical periods. Theodor Mommsen's prosopography is typically considered the first prosopography and listed all position holders of the Roman Empire. The first prosopographies were little more than a list of names and associated background information such as year and place of birth, relatives and important positions. It was not until the second half of the 20th century that prosopography was lifted from its purely descriptive role into a more explanatory task. Lawrence Stone envisaged a prosopography that would serve to explain political action, to capture ideological and cultural change, and to describe and analyse the structure of society [1]. In the years that followed, prosopography regained ground not only in historiography but also in sociology and political science. Especially in French-language literature, prosopography is still used to -through the study of elite trajectories- reveal something about society [2]. In the English-language literature, career trajectories are often studied for similar reasons albeit not necessarily under the heading of "prosopography".

Despite the various purposes for which prosopography or the study of career trajectories is employed, the method, or at least its first step, remains essentially just the same as Theodor Mommsen's first prosopography: : compiling a list of individuals belonging to a well-defined political elite and collecting as much biographical data on them as possible. Unfortunately, we noticed in many cases that the means of collecting these data is not very different from 19th century practices either. We find that data collection on elites often faces the same problems. First, there are problems with the sources themselves. In many cases, governments or companies are not too eager to share data on their elites or manage their data in a poor way. Thus, data on political elites is a scarce resource. Unfortunately, in the cases where academia does manage to get its hands on data, we find that here too, the management of this data often leaves much to be desired. Data management often consists of preparing an excel file in which one or more researchers work together that is stored on a local machine. When the researcher finishes the study and leaves the institution, the data often disappears along with it. At best, when the data does survive, it must be discoverable by other researchers and the coding of the variables must be sufficiently documented to be useful. Attempts have been made in the past to counter these problems with software solutions. [3] name some attempts to encompass prosopography with software packages but conclude that most of these solutions make too many assumptions about the shape of

⋆e-mail: tom.bellens@kuleuven.be

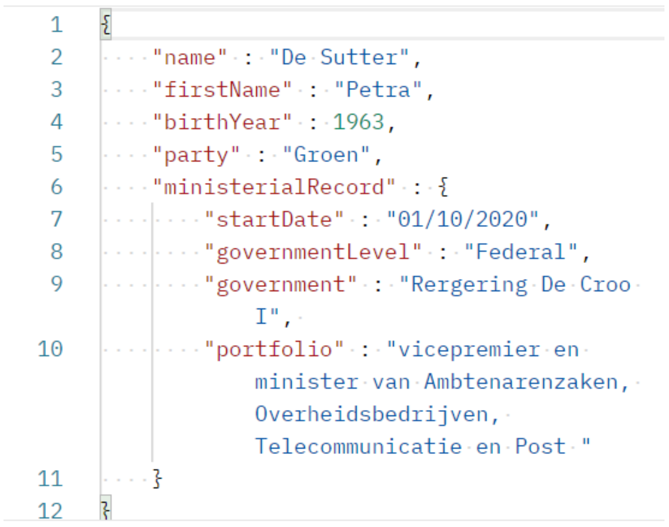
the data and are therefore not widely used.

The software package we will put together attempts to solve the problems mentioned above but will try to do so in a way that makes no assumptions about the shape of the data. Specifically, the source code of the software package is generated each time using a JSON file that defines the shape of the data. The generated software package then will always consist of a PostgreSQL database, a Java Spring Boot back-end / REST service and an Angular front-end in which data points can be created, accessed, updated and deleted. First, we will delve a little deeper into what the standard software package would look like. Then we will discuss how JSON files can be used to form the source code of the software package. Finally, we briefly discuss how these JSON files can be used to arrive at a prosopographical repository.

3 The default software package

The final prosopographic software package should a) save time and precision b) promote collaboration between staff on the same prosopographical project and c) be flexible to the needs of the specific project. As just mentioned, three components will be generated for each software package. A PostgreSQL database, a back-end in Spring Boot and a front-end in Angular. This architecture was chosen because it allows end-users without any knowledge of Java, Javascript or SQL to have a fully finished and usable software package, but at the same time allows end-users who do have some knowledge on one of these frameworks to easily adapt the source code to their own needs. That would be possible in several ways. An end-user could use the generated PostgreSQL and Spring Boot code and then use it as a REST-service from their own application / front-end. Another end-user could use the generated back-end as a starting point and implement their own functionalities to transform the data or connect to one of their own REST-services.

However, a user who does not modify anything in the source code and deploys it as is will already find a useful data processing package. The basic functionality on the Angular front-end consists of the data input forms, the overview of the data and the edit tab, whose fields are each generated from the initial JSON file (see next section). It also offers functionality to map and export existing .csv files to the software package and functionality that automatically scrapes Linkedin CVs and adds them to the data. Both manual data entry and input via Excel or Linkedin take into account pre-existing data. If a particular Linkedin CV contains data on a person who may have been added previously, it is automatically suggested to check the old entry and merge it if desired. Finally, the Spring-Boot / Angular architecture also offers the advantage that there are already very many boiler-plate libraries that make collaboration between different users easy. There will always be an "administrator" who can create and add new users and determine which operations these users can or cannot perform. A log will also be kept of the actions so that it will always be clear who added, edited or deleted which data points at what time. Any action in this log can be reversed at any time.



```
{
    "name" : "De Sutter",
    "firstName" : "Petra",
    "birthYear" : 1963,
    "party" : "Groen",
    "ministerialRecord" : {
        "startDate" : "01/10/2020",
        "governmentLevel" : "Federal",
        "government" : "Rergering De Croo I",
        "portfolio" : "vicepremier en minister van Ambtenarenzaken, Overheidsbedrijven, Telecommunicatie en Post "
    }
}
```

Figure 1. An example JSON file

4 JSON-based source code generation

As mentioned earlier, flexibility was a feature central to the design of this software package. Recognising that prosopographical data can take many different forms, software packages are always generated based on the data envisioned by the end user. Specifically, this means that the end user sets up one (imaginary) prosopographical individual in a JSON file, passes this JSON file to our compiler [1] and it generates the PostgreSQL, Spring Boot and Angular source code which can then be deployed. We briefly discuss each step below.

First of all, a JSON file needs to be created that models one individual that will serve as a template for all individuals that will be collected in the prosopography. This means that all the variables the researcher wants to collect must be present in this file. For instance, if the end-user wants to do a prosopographical study on ministers, they may be interested in the following variables: name, first name, date of birth, party, and their ministerial record. Figure 1 is an example on what such a JSON file might look like. Please note that the variable names (in camelCase) are important for generating the source code but the contents of the fields are less so. In this example, we entered Belgian minister Petra De Sutter but we could just as easily have entered another minister or a completely fictitious one. Giving the fields a content is only important for the compiler to know which kind of variable needs to be added (String, Integer, List,...). Secondly, the variable "*ministerialRecord*" should be referenced. It contains another nested JSON object. This is one of the reasons why we chose to have the data design

[1] for lack of a better name we called it *ProsopoComp* for now

done in JSON. It is an accessible format that can also be used by non-technical users that at the same time allows for hierarchical data shaping (as prosopographical data often is).

The compiler then, is a relatively simple Python script that converts the JSON file to the required source code via a few intermediate steps. The first task of the compiler is linearisation of the hierarchical data. This means mapping the data to a sequence of instructions that respects the shape of the data. Specifically, depending on how many nested JSON objects there are, one or more .pit files are created. These .pit files contain information about the variables in the JSON objects and can be read by both the script and humans as a sequential representation of the data. Such a sequence could look like this:

> "CREATE minister;ADD str name;ADD str firstName;ADD* ministerialRecord ministerialRecord;TERMINATE".

Once the data is linearised with the .pit files as a result, the compiler enters the second phase: the generation of the PostgreSQL, Spring Boot and Angular code. This second phase varies depending on the framework for which the code is to be generated but is always a mapping from a .pit file to the target language. Broadly speaking, it could be said that the above example could be mapped from .pit to Java Spring Boot by first creating a Java class "Minister", and creating the fields "String name", "String firstName", etc... in it. Then the constructors and getters and setters can be created for these variables. This class would then be stored in a file Minister.java. For a Spring Boot REST service, a data-transfer object (DTO) must then of course also be created for this minister class and functions must be provided in the Controller that can receive and emit such a DTO but again, automation based on the .pit file is not an overly complex task. Similar examples could be given for the generation of PostgreSQL and Angular code.

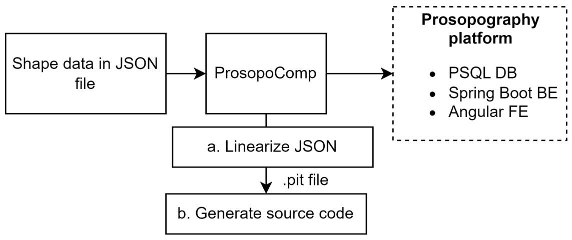


Figure 2. JSON-based source code generation

5 A repository for all prosopographical projects?

This brief summary of the software we are trying to create is not yet complete and possibly even unclear. So we welcome any feedback, both technical and in terms of functionality. Despite the title, this paper is in reality about two pieces of software. On the one hand, the "default" software package that will eventually be used by the researcher(s) and, on the other hand, the compiler that generates these software packages. The aim is to make prosopographical data collection more efficient, sustainable and to simplify collaboration. This way of working is only one of many ways to achieve this but we think that our architecture, both that of the compiler based on JSON files and that of the default software package, meets as many of the predefined requirements as possible. Finally, in our wildest dreams, there is an additional feature that emerges from this way of working. In case several researchers carry out their prosopographic research with one of our generated software packages, all the JSON files used for generation can be kept in a central repository. These files can then additionally be used as meta-documentation by users on other prosopographical projects. In this way, our software package would promote collaboration not only within research teams but also between different teams working on a prosopographical project.

Acknowledgements

Supported by FWO (grant number G054121N).

References

[1] L. Stone, Daedalus **100**, 46 (1971)
[2] P.M. Delpu, Hypothèses **18**, 263 (2015)
[3] K. Keats-Rohan, *Prosopography* (SAGE Publications Ltd, London, 2019), ISBN 978-1473965003