Public Health and Informatics
J. Mantas et al. (Eds.)
© 2021 European Federation for Medical Informatics (EFMI) and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/SHTI210155

MainzelHandler: A Library for a Simple Integration and Usage of the Mainzelliste

Daniel PRECIADO-MARQUEZ[a], Ludger BECKER[a], Michael STORCK[b], Leonard GREULICH[b], Martin DUGAS[b] and Tobias J. BRIX[b,1]
[a]*Institute of Computer Science, University of Münster, Germany*
[b]*Institute of Medical Informatics, University of Münster, Germany*

Abstract. Pseudonymization plays a vital role in medical research. In Germany, the Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V. (TMF) has developed guidelines on how to create pseudonyms and how to handle personally identifiable information (PII) during this process. An open-source implementation of a pseudonymization service following these guidelines and therefore recommended by the TMF is the so-called "Mainzelliste". This web application supports a REST-API for (de-) pseudonymization. For security reasons, a complex session and tokening mechanism for each (de-) pseudonymization is required and a careful interaction between front- and backend to ensure a correct handling of PII. The objective of this work is the development of a library to simplify the integration and usage of the Mainzelliste's API in a TMF conform way. The frontend library uses JavaScript while the backend component is based on Java with an optional Spring Boot extension. The library is available under MIT open-source license from https://github.com/DanielPreciado-Marquez/MainzelHandler.

Keywords. Pseudonymization, Open-Source, Mainzelliste, Spring Boot, JavaScript

1. Introduction

In clinical trials, patient data should be gathered in a pseudonymized fashion to protect the subject's identity as stated by the European Medicines Agency [1]. This means, that a unique identifier, i.e., pseudonym, should replace all personally identifiable information (PII). For multi-center studies, this requires a single, centralized pseudonymization service, which ensures, that identical patients at different centers are always associated with identical pseudonyms. In Germany, the Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V. (TMF) has published a guideline on how pseudonymization services should create and handle pseudonyms [2]. An open-source pseudonymization service implementing this guideline and therefore recommended by the TMF is developed by the University of Mainz, the so-called "Mainzelliste" [3]. It is used, e.g., by a register for multiple sclerosis with 43 centers and by the market leading bio-banking software in Germany [4,5]. The Mainzelliste provides a REST-API for the (de-) pseudonymization of patients. For security reasons, these calls require a complex session and tokening mechanism and a specific interaction between front- and backend of the application using this API. The backend may only know the

[1] Corresponding Author, Tobias J. Brix, Institute of Medical Informatics, University of Münster, Albert-Schweitzer-Campus 1, Building A11, 48149 Münster, Germany; E-mail: tobias.brix@uni-muenster.de.

application key, while the entered PII must stay with the frontend. Objective of this work is the development of an open-source library for simple integration and usage of (de-) pseudonymization in web applications based on Java. The library consists of a JavaScript implementation handling all frontend interactions and a backend counterpart. The backend library is divided in a core Java component and an extension dependent on Java Spring Boot.

2. Methods

2.1. Workflow of (De-) Pseudonymization with the Mainzelliste

The TMF conform workflow of requesting a pseudonym from the Mainzelliste is shown in Figure 1. The backend of an application uses its unique application key to start a session on the Mainzelliste. This session can be used to generate tokens for the actions the client, i.e. frontend, wants to perform; in our case a pseudonymization. The actions that can be requested are based on the privileges associated with the application key of the application. Thus, the key should be kept secret and not being exposed in the frontend. A token is typically limited to a single use and expires immediately afterwards. However, in the latest version 1.9 of the Mainzelliste the number of uses per token can be configured. Nonetheless, it is always possible to request multiple tokens in a single session. By default, first name, last name, and date of birth are used for record linkage, but the data fields can be configured within the Mainzelliste [6]. If no exact match is found, but the similarity of records is very high, indicating a typing error, the Mainzelliste warns the user and requests a review of the committed PII (sureness request). The workflow of depseudonymization is similar. More details are provided in the API documentation of the Mainzelliste [7].

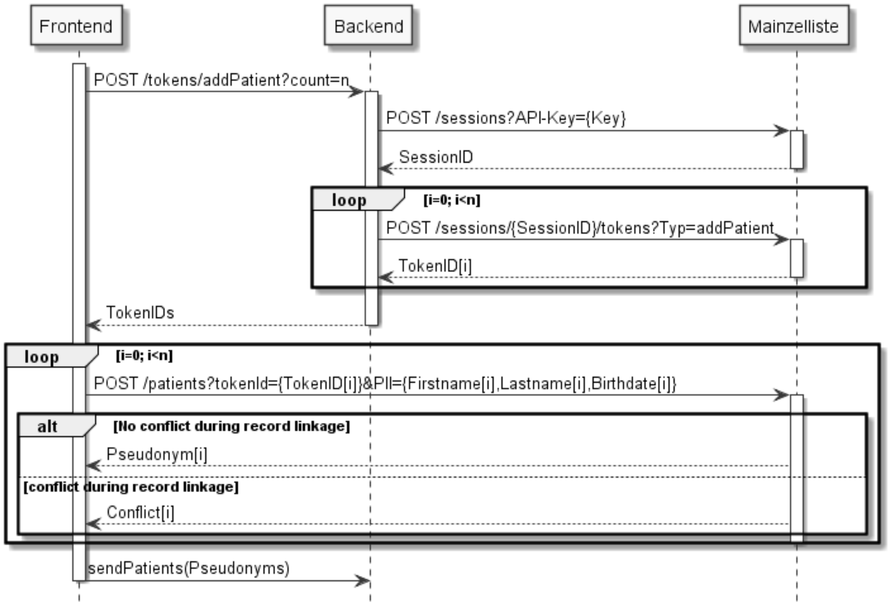
Frontend
Backend
Mainzelliste
POST /tokens/addPatient?count=n
POST /sessions?API-Key={Key}
SessionID
loop
[i=0; i<n]
POST /sessions/{SessionID}/tokens?Typ=addPatient
TokenID[i]
TokenIDs
loop
[i=0; i<n]
POST /patients?tokenId={TokenID[i]}&PII={Firstname[i],Lastname[i],Birthdate[i]}
alt
[No conflict during record linkage]
Pseudonym[i]
[conflict during record linkage]
Conflict[i]
sendPatients(Pseudonyms)

Figure 1. Workflow of requesting a pseudonym from the Mainzelliste.

2.2. Predefined Library Requirements

The backend implementation should support Java Spring Boot since all web applications developed by the developing institute are based on this framework. The frontend implementation should be done in JavaScript since it is a natural choice in web development and has synergies with the Spring Framework. According to the TMF, the PII of a patient should not be sent to the backend but rather be sent directly from the client to the pseudonymization service. Thereby, the PII cannot be logged by the application server, which should only process pseudonymized data. This restriction must be fulfilled by the frontend implementation. The operations de- and pseudonymization should be implemented. In addition, the sureness request of misspelled PII should be supported. Finally, a batch (de-) pseudonymization should be provided. Thus, lists of patients can be processed in a single request. This can be useful, for instance, if during a migration process, e.g., change of pseudonymization service, all existing pseudonyms must be re-pseudonymized using the new service. Of course, the integration and usage of the library in existing and newly developed applications should be straightforward.

3. Results

The source code of the library is available from https://github.com/DanielPreciado-Marquez/MainzelHandler under MIT open-source license.

3.1. Frontend and Backend Library Implementation

The entire architecture of the MainzelHandler can be seen in Figure 2. The frontend library implementation is written in JavaScript. It provides all previously defined functionality. The library defines a generic class “Patient” with functions to create and modify its objects. A patient object consists of PII and a string representing any arbitrarily formatted medical data. The original workflow depicted in Figure 1 is now simplified to a single JavaScript call “sendPatients(…)”. The session and token handling is done internally by the front- and backend libraries and has not to be implemented individually anymore. The frontend library also handles the Mainzelliste’s response by indicating which pseudonyms have been successfully generated and for which the record linkage had high similarities and must be revaluated. If a pseudonym is successfully generated, it is transferred to the backend library accompanied by the patient’s medical data for further processing. The depseudonymization is handled analogically.

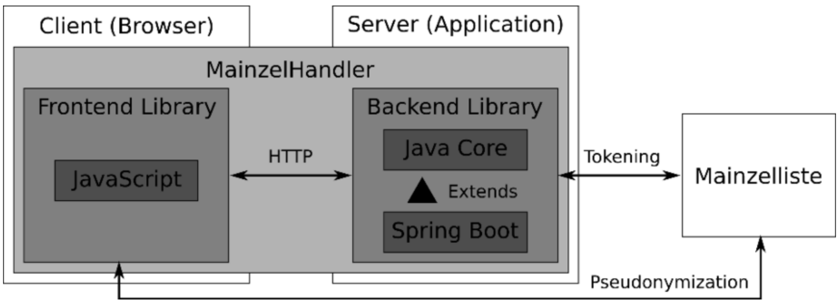

Figure 2. Architecture of the MainzelHandler.

The backend library implementation consists of a Java core component and an extension dependent on Spring Boot. Thus, applications are not forced to include the Spring dependency and may only use the core functionality. The core component manages the majority of the communication with the Mainzelliste, i.e., session and token handling, except the PII data transfer and provides a Java interface to the frontend library. However, a direct interaction between the frontend's JavaScript calls and the core component's interface is not possible, since it is highly dependent on the used application's backend framework. For Spring applications, the Spring Boot extension consisting of a single controller class implementing the core's interface is provided. The controller establishes the mapping between front- and backend library. To complete the interface, two methods must still be implemented by the application. These methods define the application's internal handling of the incoming pseudonymized patient data, e.g., storing it in a database, and cannot be predefined.

3.2. Integration and Usage of the Library

The backend library is compiled as Java Archive (JAR) file and must be included in the application. If the application is based on Spring, the Spring dependent JAR can be used and only both abstract methods of the controller class must be implemented. If another backend framework is used, only the core JAR should be included and the entire interface must be implemented. This includes the mapping to the frontend library's function calls. Independent of the included JAR, two properties have to be configured in a properties file, namely the URL of the pseudonymization service and its application key.

The frontend library can simply be included by embedding the JavaScript file. In a configuration file, the used Mainzelliste's PII fields can be defined if the defaults should not be used. An example implementation of a HTML frontend is provided as part of a demonstrator program contained in the source code.

4. Discussion

All predefined requirements of the library have been met. Its functionality has been exemplary validated as part of a Mainzelliste migration, using the batch depseudonymization and re-pseudonymization.

While pseudonymization should always be possible, depseudonymization is a critical operation. In clinical trials, a depseudonymization requires an important reason and often even an ethics vote. However, although the library is capable of depseudonymization, the authorization is done by the application key in the Mainzelliste. Thus, the functionality can be restricted without modifying the library.

Borg et al. have presented a similar client library for the Mainzelliste [8]. Their library was also developed in JavaScript for the frontend and Java for the backend. However, the intended use cases differ. Their backend has knowledge of the PII and controls which information is provided to the frontend by using so-called "tempIDs". This violates our predefined requirement that PII must not be exposed to the backend. In addition, the implementation follows closely the API definition by encapsulating each API-call and parameter in separate Java classes, which does not simplify their interaction. Furthermore, a direct connection between front- and backend is not provided and it seems to be no longer supported, since the last significant commit was in 2015.

4.1. Limitations and Future Work

A main limitation in the integration of the library in new applications is the requirement to link two libraries without a direct connection, i.e., mapping of URL calls. Thus, these mappings must still be individually implemented and are framework specific. However, for Spring applications, the mapping is already provided and reduces the implementation effort to just two functions. It should be noted that similar extensions for different frameworks could be provided in the future.

Furthermore, not all API calls of the Mainzelliste are supported yet as they were not relevant for the pivotal use case. An example is editing of an existing patient entry on the Mainzelliste. However, at the time of writing all API versions of the Mainzelliste are supported due to configurability of the version for both libraries.

As mentioned by the authors of the Mainzelliste, the generated session of the Mainzelliste can be connected to the browser session [3]. This improvement has not been implemented yet.

5. Conclusions

In this work, we presented a library for a simple usage of the pseudonymization service Mainzelliste. The front- and backend implementation can be individually included and customized. Thus, the handling of (de-) pseudonymization can be done much easier and additional functionality, like the batch import, allows quick application development.

Acknowledgements

This work has been supported by a grant from BMBF (HiGHmed 01ZZ1802V).

References

[1] European Medicines Agency. ICH E6 (R2) Good clinical practice; Available from: https://www.ema.europa.eu/en/ich-e6-r2-good-clinical-practice.
[2] Pommerening K, Drepper J, Helbing K, Ganslandt T. Leitfaden zum Datenschutz in medizinischen Forschungsprojekten: Generische Lösungen der TMF 2.0. München: Medizinisch Wissenschaftliche Verlagsgesellschaft; 2014.
[3] Lablans M, Borg A, Ückert F. A RESTful interface to pseudonymization services in modern web applications. BMC medical informatics and decision making 2015;15:2.
[4] Simbrich A, Thibaut J, Khil L, Maximov S, Wiendl H, Berger K. Chances and Challenges of Registry-Based Pharmacovigilance in Multiple Sclerosis: Lessons Learnt from the Implementation of the Multicenter REGIMS Registry. Drug safety 2020.
[5] KAIROS GmbH. CentraXX; Available from: https://www.kairos.de/produkte/centraxx/.
[6] Warnecke T, Borg A, Ückert F, Lablans M. Fehlertolerantes Record Linkage von Patientendaten durch den Phonet-Algorithmus. German Medical Science GMS Publishing House; 2013.
[7] Lablans M, Borg A, Tremper G. Schnittstelle der Mainzelliste; Available from: https://www.unimedizin-mainz.de/fileadmin/kliniken/imbei/Dokumente/MI/Mainzelliste/Dokumente/Mainzelliste__Schnittstelle.pdf.
[8] Borg A, Lablans M, Ückert F. Mainzelliste.Client – Eine Bibliothek für den Zugriff auf Patientenlisten: German Medical Science GMS Publishing House; 2015.