



# Storing Configuration Preferences using the CCDA

Project by: Massimo Hansen, *massimo.hansen@cern.ch*

Supervised by: Bartek Urbaniec, *b.urbaniec@cern.ch*

CERN, CH-1211 Geneva, Switzerland

---

---

## Contents

1	Introduction	2
2	Database Model	2
3	Client Domain and Services	3
4	Generic Configurations API	4
5	Graphical User Interface	4
6	Conclusion	6

# 1 Introduction

Storing user configuration data is definitely not a new concept. Like for example, most IDEs allow users to store customized information like run configurations used when executing code or preferences related to styling, e.g. color themes or fonts. This project revolves around implementing a solution for storing various configurations in CERN applications, and the solution should be as generic as possible to accommodate a variety of use cases. The project consists of three parts, building an API, building a client library which consumes the API, and building a GUI which makes use of the services provided in the client library. The API should be implemented as part of the CCDA (Controls Configuration Data API). The CCDA provides a RESTful API used by different applications at CERN.

The project is completed over the span of two months during the summer of 2023. This report is a short summary of the project and its results, and for further inquiries, feel free to contact me.

## 2 Database Model

The database used for the project is an Oracle Database, which is a relational database developed by Oracle Corporation.

The database model was decided before my arrival at CERN by my supervisor Bartek Urbaniec, and consists of three tables as described in the figure 1. The model describes that we have a number of generic configurations, which in turn contain a number of generic preferences. Users can then define their own values for preferences and store them as preference values in the generic preference values table. What we should understand from this structure is that the generic configurations and their preferences act as templates for which the users can then configure their own preferences.

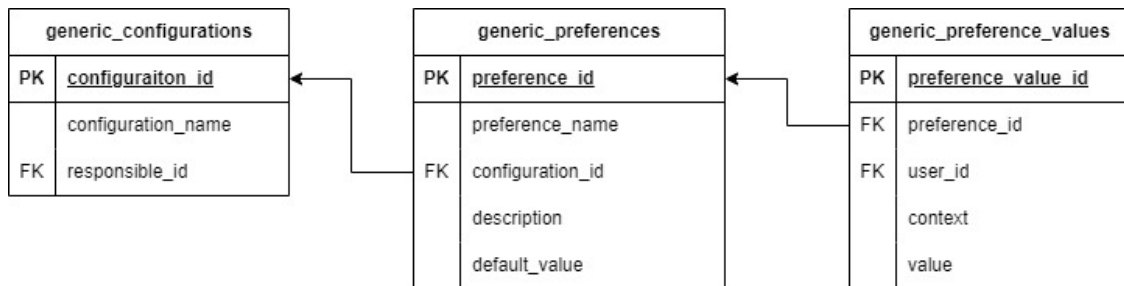


Figure 1: Simplified Logical Database Model

### 3 Client Domain and Services

The model of the client library is based off of the database model. It is written in Java and also uses Feign, a declarative web service client that allows us to implement client services by annotating interfaces.

The first thing we did was translate the entity sets from the database model into three classes: GenericConfiguration, GenericPreference and UserPreference. Here we have changed the name of generic preference values to 'user preferences' as it is a more logical name from a user perspective. We then also created a class called UserConfiguration, which isn't represented in database model. The UserConfiguration object doesn't contain any unique information, which is why it is not part of the database model. But the reason we kept it in the domain, was that we now have two main objects, generic configurations and user configurations, which then both contain their own type of preferences. The advantage of this is that we have two similar structures that need a lot of the same logic, which can then just be copied. A class diagram describing these four classes can be seen in figure 2. These

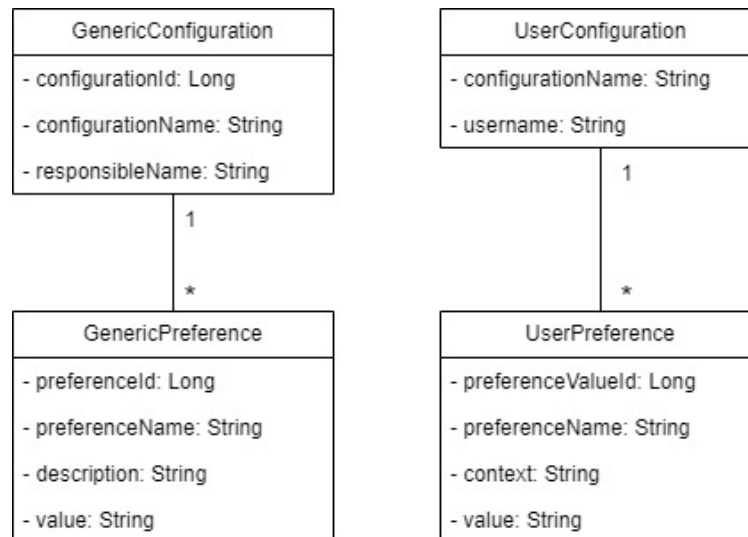


Figure 2: Simplified Class Diagram

classes make up the main package in the client library. On top of this we implemented 4 more packages, 3 of which contain similar classes as the main package but for different uses. The first contains base classes from which the classes in the main package and the 2 others inherit. The other 2 packages contains classes used as data transfer objects when sending post and put requests. The reason we created classes separately from the main package was to clarify for the user which fields are used in which cases. For example we don't to manually define and send objects with ids when doing post requests. Some classes may also contain fields which we don't want users to be able to update, and therefore we create a separate classes to send in put requests, which don't contain the fields that will never be updated.

The last package contain objects for performing bundle operations on preferences. One

might imagine, that when updating a configuration i.e. adding, updating or removing preferences, it would be useful to perform all of these changes and commit them using a single call, as it minimizes the number of requests and allows for optimized database operations.

We wrote client services using Feign, that consume endpoints provided by the CCDA, which we implemented as well. The services provide options for client to use CRUD-operations for all elements: generic- and user configurations, generic- and user preferences, as well as bundles.

## 4 Generic Configurations API

The CCDA is built using Java and Spring Boot. The generic configurations module consists of a number of controllers, services, mapper classes, authorization methods and of course some entity classes. As we previously mentioned, user configurations aren't represented by an entity set in the database, so an entity class doesn't exist for them. Instead we created a separate view in the database from which we can extract all user preferences for a specific user based on preferences from a specific generic configuration. This way we can extract all the user preferences and pass them all back to the client using a single object which contains information about the user and configuration as well.

We also implemented a simple role based access system that only consists of a single expert role. For regular authenticated users that access the API, they will only be to modify configurations and preferences they have created or for which they are responsible, while users with the expert role have access to everything.

## 5 Graphical User Interface

One of the applications that access the CCDA is the CCDE, the Controls Configuration Data Editor. As part of the project, a module also had to be built in the CCDE to allow users to create and modify both generic and user configurations. The result was four pages described as follows:

- The first page includes a table which displays all generic configuration and allows you to filter and search through them.
- The second page allows you to see details about generic configurations as well as creating and modifying their generic preferences.
- The third page is very similar to the first page, but here we display a table of user configurations that users can search and filter.
- And lastly the fourth page, similarly to the second, allows you to view details about user configurations as well as create and modify their user preferences.

In total these four pages make use of all the available functionality from the CCDA besides the bundle (batch) operations. The user interface in the CCDE is built using Angular, a

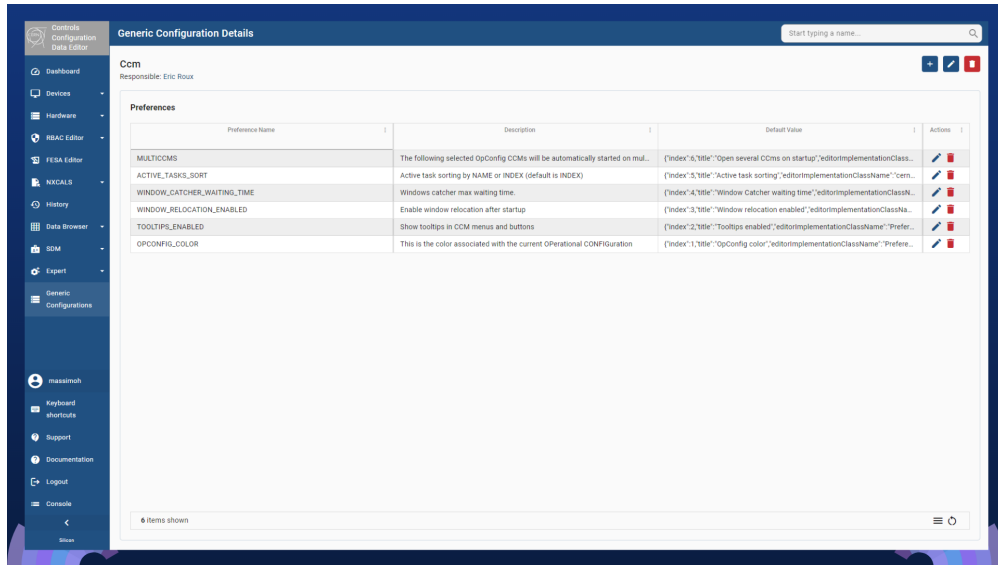


Figure 4: Screenshot of the second page: Details for Generic Configurations

framework developed and maintained by Google, which like most other frameworks used for web-development uses JavaScript/TypeScript, HTML and CSS. Below you will see screenshots from the CCDE showing the first three pages.

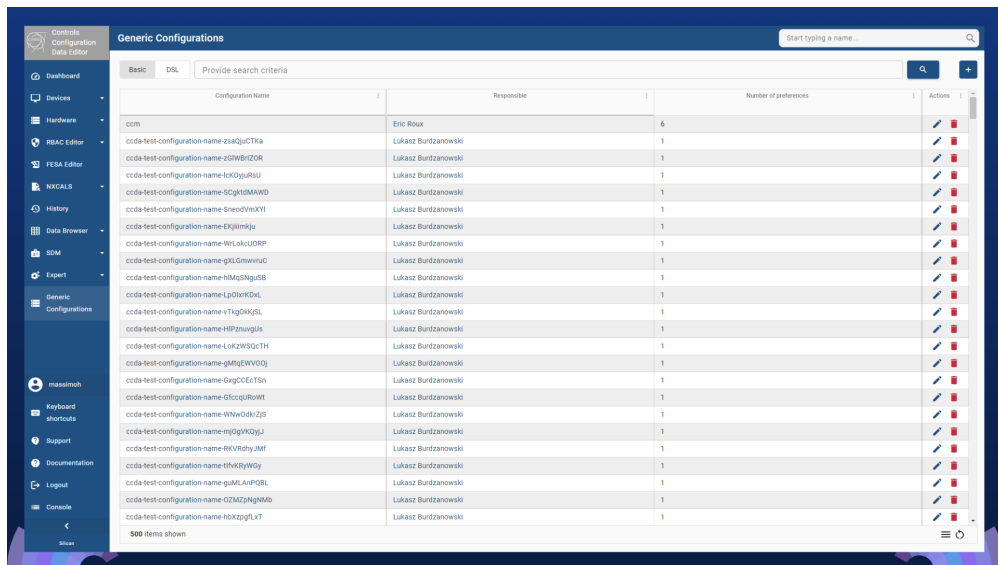


Figure 3: Screenshot of the first page: List of Generic Configurations

Owner	For configuration	Number of preferences	Actions
lbundzan	ccda-test-9kmmMhZ-upb	1	[Search] [Edit] [Delete]
lbundzan	ccda-test-lpOCgPp-eupb	1	[Search] [Edit] [Delete]
lbundzan	ccda-test-nxEgcSBa-eupb	1	[Search] [Edit] [Delete]
lbundzan	another-beautiful-configuration	1	[Search] [Edit] [Delete]
lbundzan	ccda-test-DNnUhiqX-upb	1	[Search] [Edit] [Delete]
lbundzan	ccda-test-UaznABOn-upb	1	[Search] [Edit] [Delete]
lbundzan	ccda-test-WCaggoWUr-upb	1	[Search] [Edit] [Delete]
eroux	ccda-test-lpOCgPp-eupb	1	[Search] [Edit] [Delete]
eroux	ccda-test-nxEgcSBa-eupb	1	[Search] [Edit] [Delete]

Figure 5: Screenshot of the third page: List of User Configurations

## 6 Conclusion

In the end most goals for the project were reached, as we managed to implement all three parts: client library, API and front-end. Of course all parts (especially the work done in the CCDE) need extra work and will be subject to refactoring and feedback from users. Before the start of the project, I had experience with many of the technologies used in the project, but I still faced many challenges. Especially using Angular for the front-end was a challenge, as I had never used the framework in previous projects.