Date: 22.06.2021



Exemplar Management System

Team 04

Kevin Schütz, Valentina Hummenberger, Julia Hammer

Project  
documentation

Table of Contents

[1. Introduction 3](#_Toc75538606)

[2. Implemented Requirements 3](#_Toc75538607)

[3. Overview of the system from the user point of view 3](#_Toc75538608)

[4. Overview of the system from the developer point of view 15](#_Toc75538609)

[4.1. Design 15](#_Toc75538610)

[4.1.1. Overview of the system 15](#_Toc75538611)

[4.1.2. Important Design Decision 15](#_Toc75538612)

[4.2. Implementation 16](#_Toc75538613)

[4.3. Code Quality 16](#_Toc75538614)

[4.4. Testing 23](#_Toc75538615)

[5. Installation instruction 23](#_Toc75538616)

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Creator | Changes |
| I  (Release 1) | 08.04.2021 | Kevin Schütz | Backend with Database, first implementation of User Dashboard, first implementation of Exemplar Dashboard |
| II  (Release 2) | 13.05.2021 | Valentina Hummenberger | Adaption of Libraries, Finalization of Exemplar Dashboard (e.g. comment section), changes in Database, |
| III  (Release 3) | 25.06.2021 | Julia Hammer | Implementation of Communities and the according library, finalization of Database (hosted locally), |

Table : Version history

# Introduction

We implemented an Exemplar Management Tool as a desktop application. This tool enables Users to log in, join communities and view, rate, download or label exemplars. If they care to be creators, they must confirm accordingly whilst registration or opt in later. If a user is also declared a creator, she/he is enabled to share her/his exemplars. An exemplar is hereby a program designed by a creator, which tries to solve a specific Problem.

# Implemented Requirements

We implemented all the requirements pointed out in Moodle, except the optional requirement of the most accessed exemplar. We did not exactly contribute each of the requirements to a team member, but programmed together and finished where another member started, when indicated. If the requirements must be parted amongst the team members, we would conclude, that Kevin was responsible for the basic requirements except the Communities and the Comment section. Valentina was responsible for adding the comment section and the Analysis part. Whilst Julia did program the Communities and was responsible for the sort sections.

# Overview of the system from the user point of view

The system is used as a local application, connected with a data base. One enters the application via the desktop icon and is asked to log in as follows:



1. Basic: Create/Retrieve/Update/Delete an Exemplar profile (Name, Contributors, Context, Problem, Solution,...)

Create: At the Hometab exemplars can be created if one chooses to be a creator:



Retrieve: An exemplar can be accessed at the homepage (see picture above), the Exemplar library, or the search function:



Update/Delete: In the exemplar dashboard an exemplar can be updated as well as deleted



2. Basic: Create/Retrieve/Update/Delete a User profile

Create: When one opens the application and does not yet have a User profile, there is the possibility to “register” and create a User profile:



Retrieve/Update/Delete:

When logged in the User Profile can always be retrieved, updated and deleted in the Home Tab:



3. Basic: Create/Retrieve/Update/Delete a Contributor profile. A Contributor is a registered User

We implemented the Contributor profile as an extended User profile. If one decides to contribute, the Button “Contributor?” can the chosen in the Registration form. There is also the possibility to opt later in the User profile. The basic process is already explained in requirement number 2. Furthermore there is the possibility to search for creators:



Creators can also be accessed via the Contributor library:



4. Basic: Label can be assigned to Exemplars by Users

In the Exemplar dashboard/ Exemplar tab any label can be assigned through clicking a button at the bottom and giving a chosen name:



5. Basic: Ratings can be assigned to Exemplars by Users

Ratings can be assigned quite similar as requirement 4 in the exemplar tab:



6. Basic: Create/Retrieve/Update/Delete communities of users. Each community contains a list of reference exemplars.

Create: In the Home Tab a new community can be created, if the name differs from any existing community:



Retrieve/Update/Delete: Through clicking “open selected” one can choose which community to retrieve and Update or Delete in the opened tab:



Furthermore there is the possibility to search for communities:



7. Basic: Users can comment and reply to comments about an Exemplar

By clicking “Leave Comment” in the exemplar section any comment can be given:



Also in the exemplar dashboard, any user can reply to a comment:



8. Queries: Show exemplars with specific labels attached.

In the exemplar library by marking “Filter by label” one can choose exemplars accordingly:

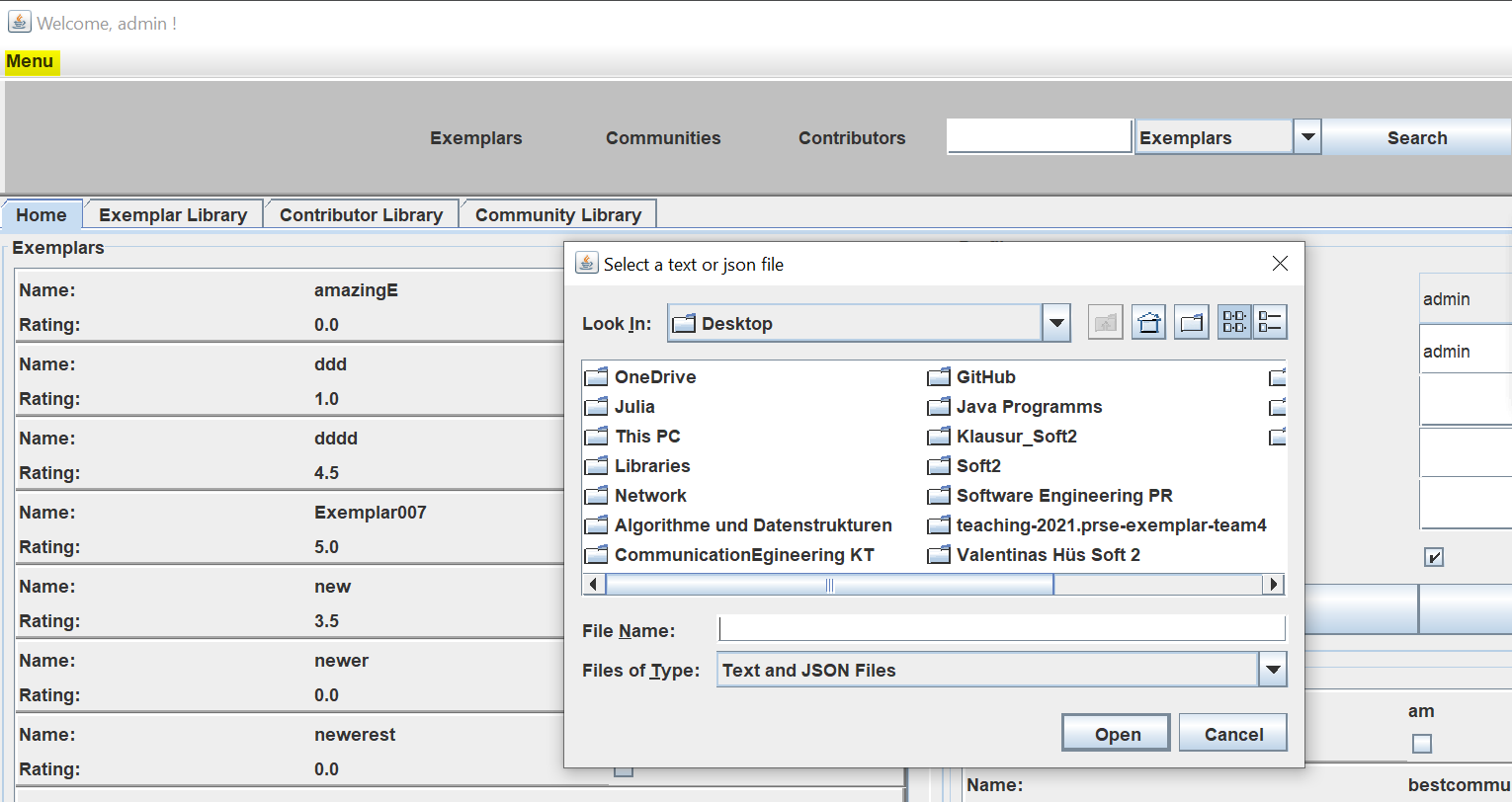


9. Import/Export: JSON-based import/export of Exemplars

In the exemplar tab through clicking on “export” a specific exemplar can be stored locally:



Import: Menu – Exemplars – Import:



10. Analyse: Exemplar Dashboard including contributors, users, labels, and ratings.

The exemplar dashboard can be accessed through the home tab, the exemplar library, or the “search” function. Here is a picture of our Exemplar dashboard, which contains the contributors, labels and ratings:



11. Analyse: Contributor Dashboard including contributed Exemplars, labels of contributed Exemplars, and overall rating of contributed Exemplars.

Out contributor dashboard can be accessed through the contributor library or the search function. It contains the contributed Exemplars, the labels of the exemplars an the overall rating of each exemplar plus an average rating over alle exemplars regarding the contributor:



12. Analyse: Trend analysis - Which are the most (accessed and) rated exemplars of the last week? () = optional requirement

In the exemplar library the most rated exemplars can the accessed by choosing the according button:



13. Sort: Classify Exemplars by avg. rating, by # of users

In the exemplar library the sorting by rating can be chosen:



14. Sort: Classify Top Contributors w.r.t. a particular label

In the contributor library contributors can be sorted according to a particular label:



15. Sort: Classify Top Contributors w.r.t. ratings of contributed Exemplars

In the contributor library the sorting by the average rating of an exemplar can be chosen:



# Overview of the system from the developer point of view

## Design

### Overview of the system

For the desktop application we tried to stick to the Model-View-Controller principle as far as possible, so we tried to separate the business logic from the view using controller-classes and from the model by separate Http-Clients which access the backend for manipulating the data in the SQL database.

Our general architecture follows a 3-layered architecture, where the client is the desktop- application and the server is our Spring Boot application. Our model is represented in the database.

For the UML diagrams, see github branch.

### Important Design Decision

*Description of the 3-5 most important design decisions according to the following scheme*

**Decision:** Contributors are users, who must declare, that they want to contribute at registration or opt later.

**Reason:** Since the requirements did not conclude how to implement creators specifically, we had to decide.

**Alternatives that were considered:** An extra log-in frame for contributors without the possibility to opt.

**Assumption:** We assumed that our implementation would be easier for users to handle and that, with the given possibility to opt, we might win more contributors. We assumed furthermore, that the more contributors there are, the better it will be for our program and communities.

**Consequences:** A somewhat lighter program missing another extra feature, making the handling easier for the users. Possibly there might be more contributors.

**Decision:** How to add the exemplars to the communities. We decided, that each exemplar shall be added to a specific community separately.

**Reason:** It was not defined, how communities shall work specifically. The Question was, whether communities focus on exemplars or users.

**Alternatives that were considered:** The alternative was to focus on the users and add alle exemplars of every user to the community.

**Assumption:** Since the focus of the exemplar management tool is on the exemplars rather than the users. We assumed that communities would be formed, not to boast whichever community contributed more exemplars and stash them, but to work together. We reasoned, that one might also want to add an exemplar from a contributor, not part of the community, to help with the community problems.

**Consequences:** Therefore we implemented, that every exemplar available can be added to any community. If a user/creator joins a community, his/her exemplars are NOT added to the communities exemplars automatically.

**Decision:** Menu Panel with libraries

**Reason**: At first, we did not have a menu panel on top. However, when the implementation process proceeded, the program seemed to be less user friendly and more confusing to handle.

**Alternatives that were considered:** Adding the libraries to the Home Tab.

Assumption: We assumed, that our implementation with the menu panel would be more user friendly. Since the menu can be addressed with any panel open, users are more likely to find it.

**Consequences:** Buttons on top and on the bottom of the program were added.

## Implementation

Frontend-wise we divided our project into three areas: the model itself, which holds all the relevant data, the views, which include the design elements and finally the controller, which implements most of the functionality of the application.

We created entities for Exemplars, Users, Ratings, Labels, Comments and Communities.

Corresponding to these entities we implemented separate HTTPClients for Exemplars, Users, Ratings and so on. These are responsible for communicating with our database and thus, for fetching data. In order to be able to execute sorting and filtering operations more smoothly, we also used streams whenever it was possible (java.util.stream Library). Our Controller consists of the LoginController, which is – as the name suggests – responsible for the login in operation, and the MainController, which provides the main functionality of our application. Regarding the user interface we heavily relied on the Swing library. We implemented the user interface by creating different frames, panels, listeners and events. Tabs play an essential role in our implementation since the user can access most of the information via tabs (= panels which are integrated into the main frame).The most important panels include the Home Panel, the Exemplar and Contributor Dashboard and the Library Panels. Most of the important user operations can be carried out via buttons (for instance creating a new Exemplar or closing a tab). However, we also used JCheckboxes and JComboboxes, for exemplar in connection to the filtering operations in the different libraries. In addition, we also used a MenuBar in order to disply the existing libraries in the main frame.

Our Backend is implemented as a Spring Boot application and is responsible for proxying the database requests between the Frontend and our Microsoft SQL Database. For this we used the Java Persistence API with Hibernate. This requires us to have all the necessary entities of the database as Java Classes with the @Entity annotation and specific annotations to tell Hibenate how relations, e.g when lists or maps are used, should be handled. Moreover, it is necessary to define the @ID of the entity. In addition to the entities a generic JPARepository<T> that offers all the basic CRUD-Operations as methods is needed for each entity. It is possible to define more complex or specific SQL-Queries in these repositories. Lastly, we implemented one Controller for each entity that offers the API for the Frontend. Basically, these interfaces use the repositories for manipulating the data.

## Code Quality

**Sonarlint - Code Analysis Plugin**

* Issues bevor refactoring: 892 in 76 files
* Issues after refactoring: 60 in 22 files

**Refactored all occurrences of:**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Refactored some occurences of:**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**No refactoring of:**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Would require serious changes – maybe for the next release

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Performance not top priority – maybe for the next release

Ein Bild, das Text enthält.

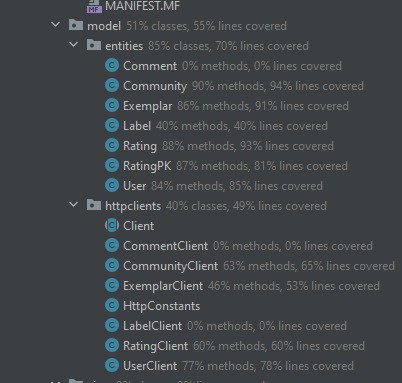
Automatisch generierte Beschreibung

For a method where we have not yet found an alternativeEin Bild, das Text enthält.

Automatisch generierte Beschreibung

Not clear how to handle them other than to print stacktracea

## Testing

**

Up until now, we have implemented 27 unit tests, with the primary focus on testing the different clients. Unfortunately, we were not completely sure about how we should test all the methods which are included in frames/panels and so on. We plan to do some more research and add more tests for the final release.

|  |  |
| --- | --- |
| Test case ID | **Test\_addUser (TestUserClient)** |
| Designed by | Kevin, Julia, Valentina |
| Executed on | 26.6.2021 |
| Carried out by | Kevin |
| Tested Requirement | Create new User/Contributor |
| Requirement |  |
| Test steps | 1. Create new Test User 2. Add User to database |
| Test data | Test User |
| Expected result | The added User should be returned |
| Postcondition | A test User must be created. |
| Status | Passed |
| Comments |  |

|  |  |
| --- | --- |
| Test case ID | **Test\_getExemplar (TestExemplarClient)** |
| Designed by | Kevin, Julia, Valentina |
| Executed on | 26.6.2021 |
| Carried out by | Kevin |
| Tested Requirement | Retrieve Exemplar by name |
| Requirement |  |
| Test steps | 1. Create Test Exemplar and add it to database. 2. Fetch Exemplar by entering the name of the previously created Exemplar |
| Test data | Test Exemplar |
| Expected result | The same Exemplar which has previously been added to database should be returned. |
| Postcondition | A test Exemplar must be created and added to the database. |
| Status | Passed |
| Comments |  |

|  |  |
| --- | --- |
| Test case ID | **Test\_addExemplar (TestExemplarClient)** |
| Designed by | Kevin, Julia, Valentina |
| Executed on | 26.6.2021 |
| Carried out by | Kevin |
| Tested Requirement | Create new Exemplar |
| Requirement |  |
| Test steps | 1. Create Test Exempar 2. Add it to the database |
| Test data | Test Exemplar |
| Expected result | The added Exemplar should be returned |
| Postcondition | A test Exemplar must be created. |
| Status | Passed |
| Comments |  |

# Installation instruction

*Description of how to install and start the system*

*(Link to github doku)*

* *Will be updated for the final release*