Università della Svizzera Italiana

Master of Science in Informatics

Software Engineering
Assignment 8

Student:

Matteo Basso

matteo.basso@usi.ch

Academic year 2019/2020

Lugano - 01/12/2019

Contents

Chapter 1

Introduction

This report summarizes the functionalities, choices, architecture and patterns that characterize the project. In this chapter general but fundamental information is provided, like the choice of the ORM library and the links to resources.

1.1 Context

The provided application implements the requirements specified by the text of the assignment, summarized here:

- Users can and must login in order to use the application.
- At least the first user must use the registration page before logging in.
- After the authentication it is possibile to perform CRUD (create, read, update, delete) operations for the entity that represents the user.
- Users can also be searched by:
  - Username
  - Name
  - Best friend
  - Country
  - City
  - Street

1.2 Hibernate

In order to develop the application it has been decided to import Hibernate [2], a popular ORM framework for Java. Using a library like this has multiple benefits in a real project, here are some of them:

- The sophisticated design of the data mapping layer is alleviated.
- It allows the usage of design patterns without implementing them from scratch. For example, Lazy loading or transactions can be used in a very simple and intuitive way.
- It handles edge cases in a better and tested way.
- Developers can think about the architecture of the system without worrying about these implementations.
- It assures the security of the system by default (for example, protecting from SQL Injection).

1.3 Resources

Code, report, diagrams and images are available in the GitHub Repository [1] of the assignment:

- Servers and implementation are available in the `/code` folder.
- Diagram sources can be found in `/code/assignment8/diagrams` and can be opened using a working installation of IntelliJ IDEA Ultimate version.
- This document is available at `/report`.
- Diagrams and screenshots are located in `/report/images`.

Chapter 2

Application

In the following chapter, a brief explanation of the different pages is provided.

2.1 Login

A login is needed in order to use the application and access its functionalities. For this reason, every user must authenticate himself as a first step. This page is shown in figure 2.1.

2.2 Registration

Users can login into the system after submitting a registration. The following page, provide a way to do so, inserting some general information. This page is shown in figure 2.2.

2.3 User list

The first page of the application shows the list of registered users. It is possible to edit and delete each of them, except the one that is logged in. It is also possible to navigate to other pages and eventually logout. This page is shown in figure 2.3.

2.4 User creation

Users can be created also by another user without using the registration page. The form is the same. This page is shown in figure 2.4.

2.5 User update

A user can also update every other using the same (precompiled) form showed before. This page is shown in figure 2.5.

2.6 User search

Through a simple view, it is possible to search users. Filters are combined only if not empty. This page is shown in figure 2.6.

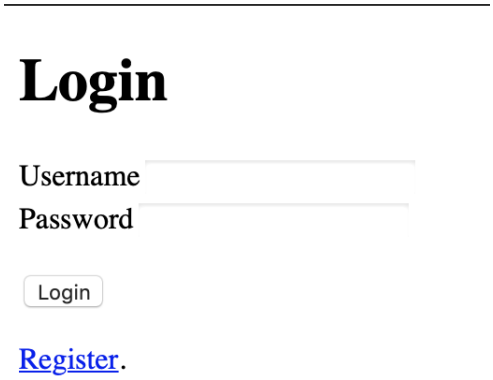

Figure 2.1: Login page

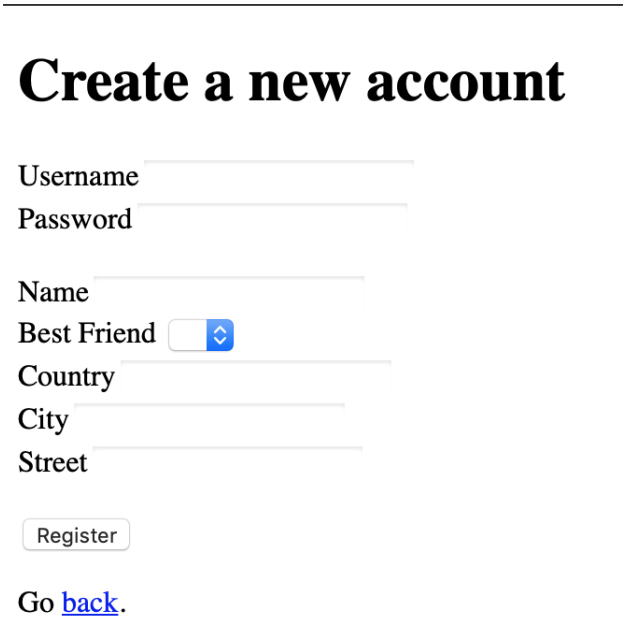

Figure 2.2: Registration page

User Management

Add New User

User list

Search user

Users

Username	Password	Name	Best Friend	Country	City	Street	Actions
User	Pass	user name	Username	Switzerland	Lugano	Via example, 70	Edit Delete
Username	password	Name		Italy	Milan	Via Pirandello, 23	Edit

Logout.

Figure 2.3: User list page

New user

Add New User

User list

Search user

Username
Password
Name
Best Friend
Country
City
Street

Add

Go back.

Logout.

Figure 2.4: New user page

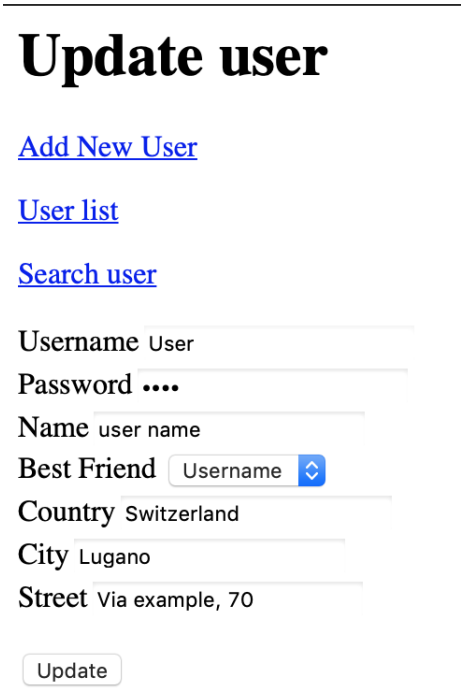

Figure 2.5: Update user page

Search user

Add New User

User list

Search user

Username
Name
Best Friend
Country
City Milan
Street

Search

Users

Username	Password	Name	Best Friend	Country	City	Street	Actions
Username	password	Name		Italy	Milan	Via Pirandello, 23	Edit

Go back.

Logout.

Figure 2.6: Search user page

Chapter 3

Design Patterns

In this chapter a short explaination of (some) used design patterns is given.

3.1 Front Controller

A single *Front Controller* that handles all requests has been implemented. At the moment it only identifies and delegates the execution to the right command. However, it will be useful in the future to perform common operations like logging, internationalization, etc without duplicating code. It also simplifies the configuration of the server.

3.2 Intercepting Filter

An *Intercepting Filter* has been used to manipulate all the requests, preventively checking if the user is authenticated and allowed to visit the requested page. He is redirected to the login page otherwise.

3.3 Template view

HTML code is generated via templates that call proper helpers to get and render data. This easy approach allows the separation of view and implementation logic.

3.4 Data Mapper

`Hibernate` uses the *Data Mapper* pattern to deal with the database. It instantiate an object for each record (row) and maps the columns with the attributes of that object. Relationships are

also represented as attributes with the type of another entity.
`Hibernate` represents the only module that use the configuration of the database, relies on a DB layer and effectively change the data.

3.5 Repository pattern

The *Respository pattern* has been used to create a layer that deals with the *Data Mapper*, encapsulating logic required to access data sources and separating it from the business one. In this way common data access functionalities are centralized and it is possible to change persistence technology in an easier way. It improves also the maintainability and decoupling of the whole software infrastructure.

3.6 Identity Field

In order to pair in-memory and DB object identities the *Identity Field* pattern has been used. In the database, the `users` table has a primary key constraint on the `username` column while in the model there is a `String` attribute that maps the `username` itself. In the case of the `Address` entity, an identity integer field representing the `id` is present.

3.7 Lazy Load

As specified in the requirements, each user might have a best friend. This is represented with an attribute in the model and translated with a relationship at the database level. When we fetch a user, it is not convenient to fetch also his best friend, since this procedure can be transitive and might end up with the fetching of the entire database (if everyone has a best friend and everyone is referenced).
For this reason, best friends can be fetched only if needed using the *Lazy Load* pattern. Fetching a single user will never cause a lot of useless allocations and data transfers.

3.8 Unit of work with Transactional Behaviour

Unit of Work can be seen into the `UserRepository` class, all changes are actuated in a transactional way using `Hibernate` APIs, even though in the given application there are no complex mechanisms to store, delete and update data.

Chapter 4

Architecture

A high-level architecture of the system is presented in this chapter. A general diagram is provided in figure 4.1 and a detailed explanation of its major subparts is provided in every subchapter. It is important to underline that diagrams are also available on the GitHub repository, to download and see them in a more comfortable way. How it has been specified in chapter 1.3.

4.1 Authentication

Authentication is managed by an *Intercepting Filter* named `AuthFilter`. It is responsible to identify the command that will handle the request (through the `FrontController`) and check the presence of an `@Authorize` annotation. Commands without the annotation can serve pages to everyone while commands with the annotation can be accessed only by logged users. This means that the filter is responsible to ensure it via the `AuthManager` and redirect to the login page otherwise.
It has been decided to use an annotation on every command to promote a declarative way of programming instead of an imperative one.

4.2 Repository pattern

A single repository is responsible to perform CRUD operations on the database using `Hibernate` APIs, as we can see in figure 4.3. It can be noticed that certain commands call directly the repository while others instantiate helpers that will be later provided to the views. Firsts are usually endpoints of HTML forms that must perform create/update/delete actions while seconds are usually commands that must show information. It is certainly possible to create helpers also

to perform the first kind of operation but that logic should be kept well isolated from the view.

4.3 Views and Helpers

As explained in chapter 3.3 the application uses the *Template View* pattern to generate HTML code. JSP files call methods defined in helper classes to get and render data. These helpers are able to call the repository to interact and perform operations on the database. Obviously, models are then used by every class. The corresponding diagram is shown in figure 4.4.

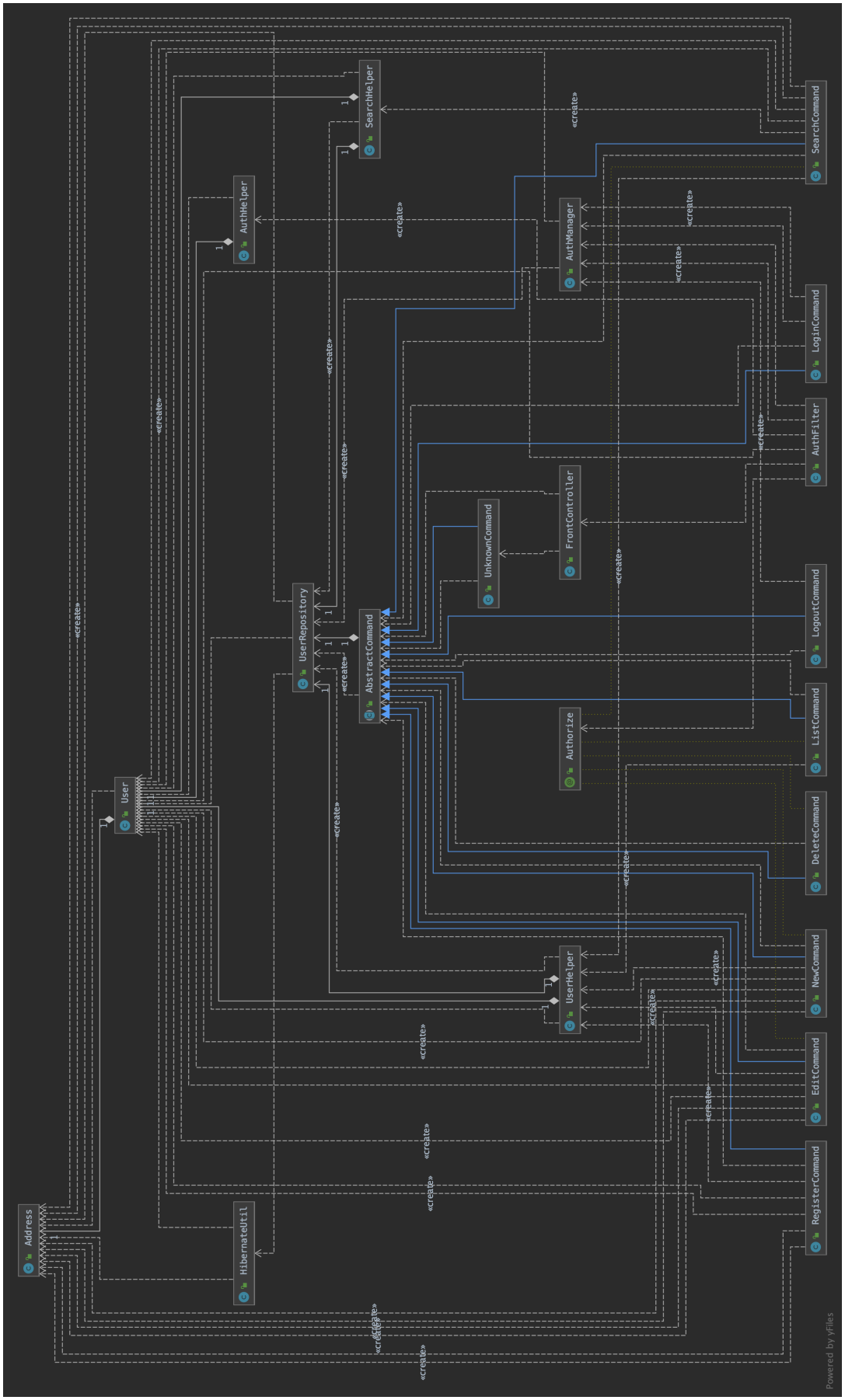

Figure 4.1: General diagram

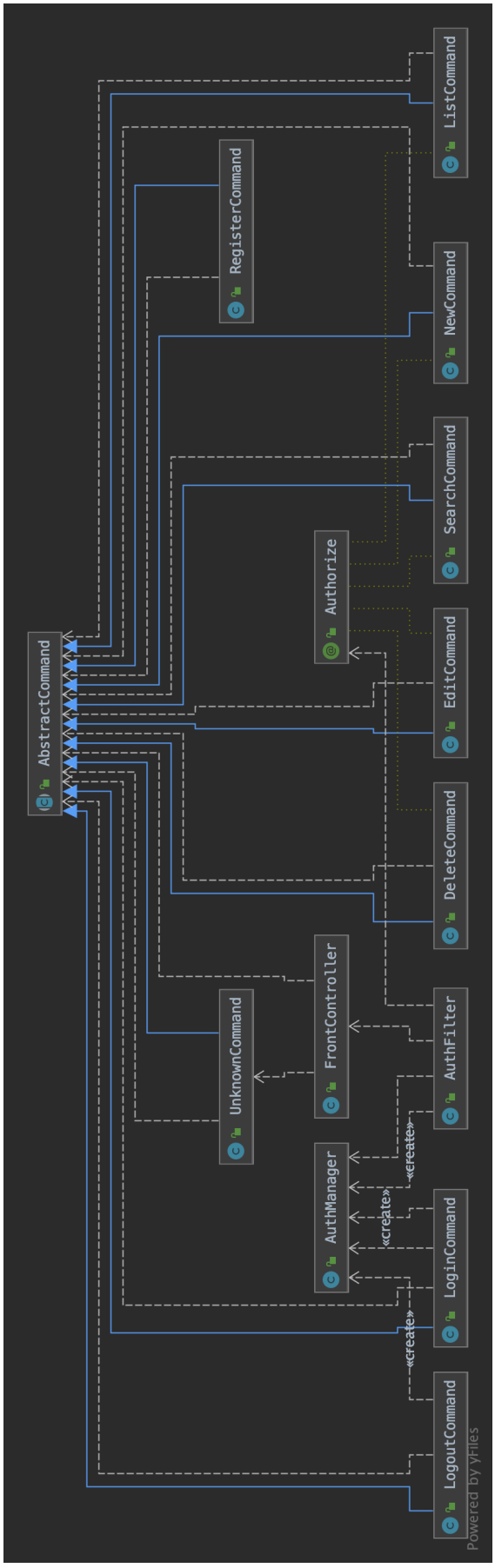

Figure 4.2: Authentication diagram

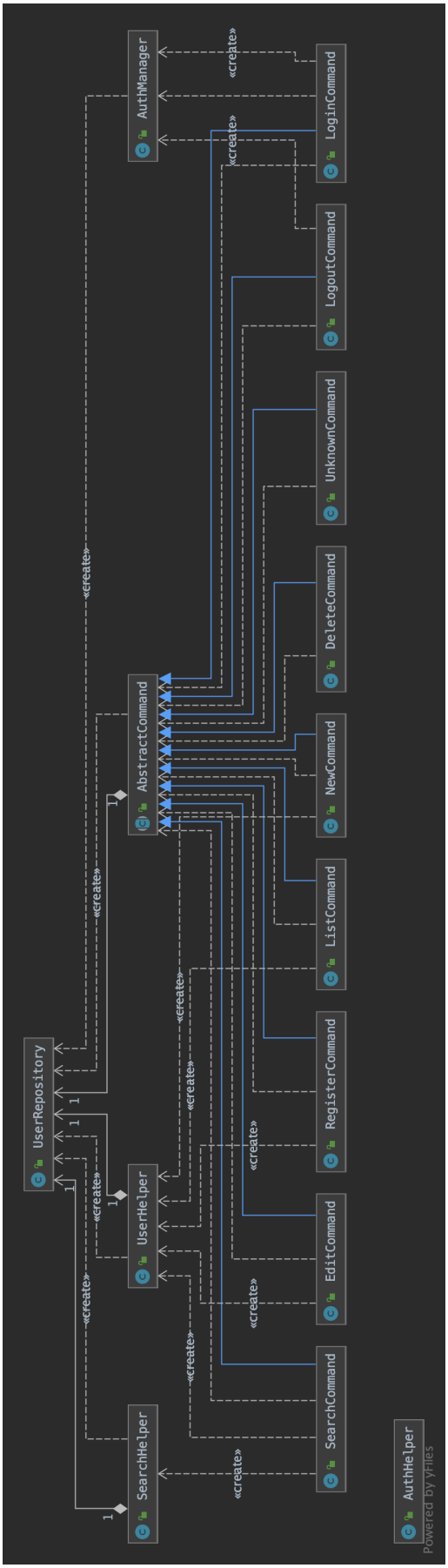

Figure 4.3: Repository diagram

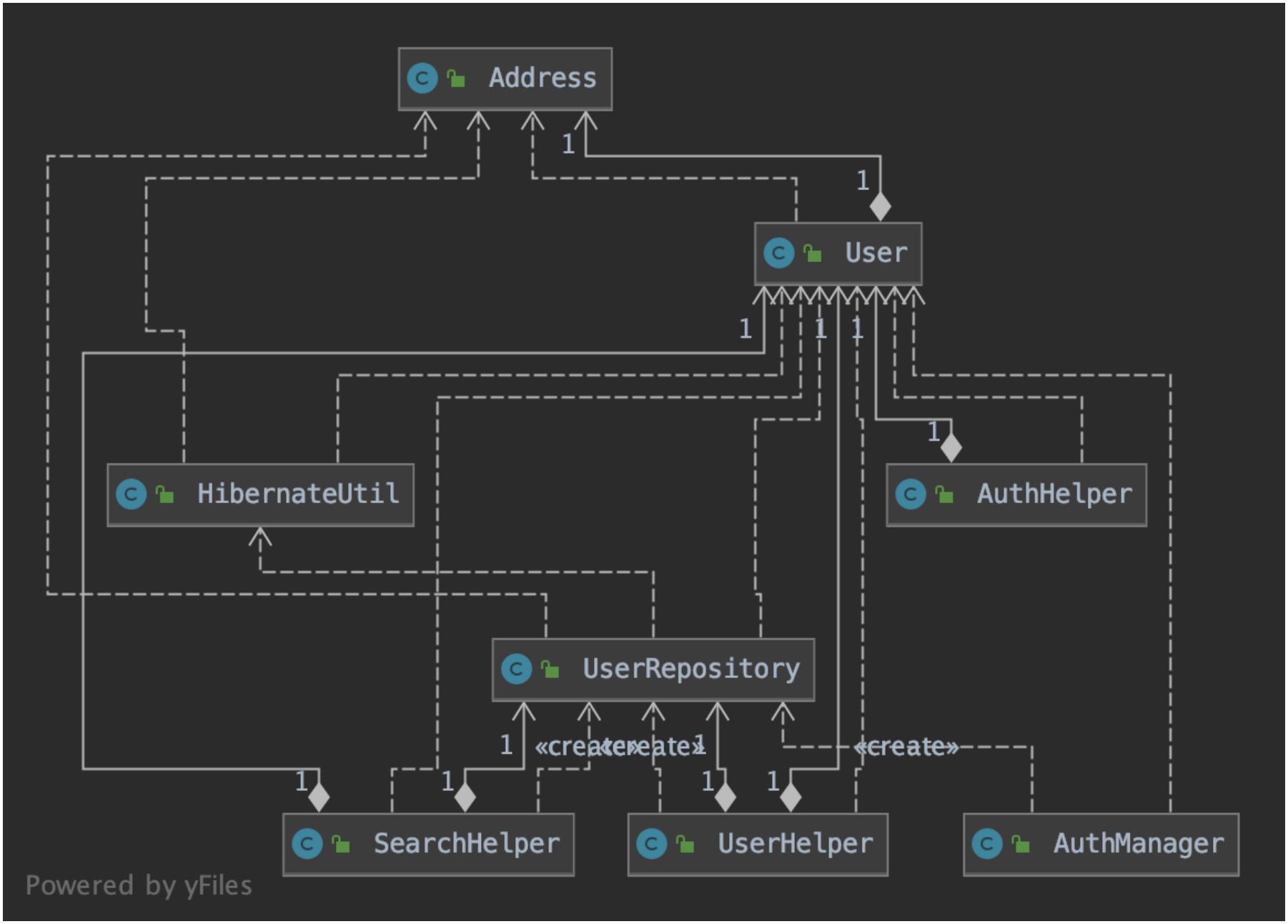

Figure 4.4: Helpers diagram

Bibliography

[1] *Assignment 8 repository*. URL: https://github.com/mbasso/se-assignment-8.

[2] *Hibernate*. URL: https://hibernate.org/orm/.