The purpose of this documentation is to carefully describe the Data Access Object (DAO) design pattern, which was chosen in order to improve the functionality of the website-database application project. First of all let us describe how does the DAO pattern work, where and why it Is used for.

The Data Access Object (DAO) pattern is a structural pattern that allows us to **isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API**.

The functionality of this API is to hide from the application all the complexities involved in performing CRUD operations in the underlying storage mechanism. This permits both layers to evolve separately without knowing anything about each other.

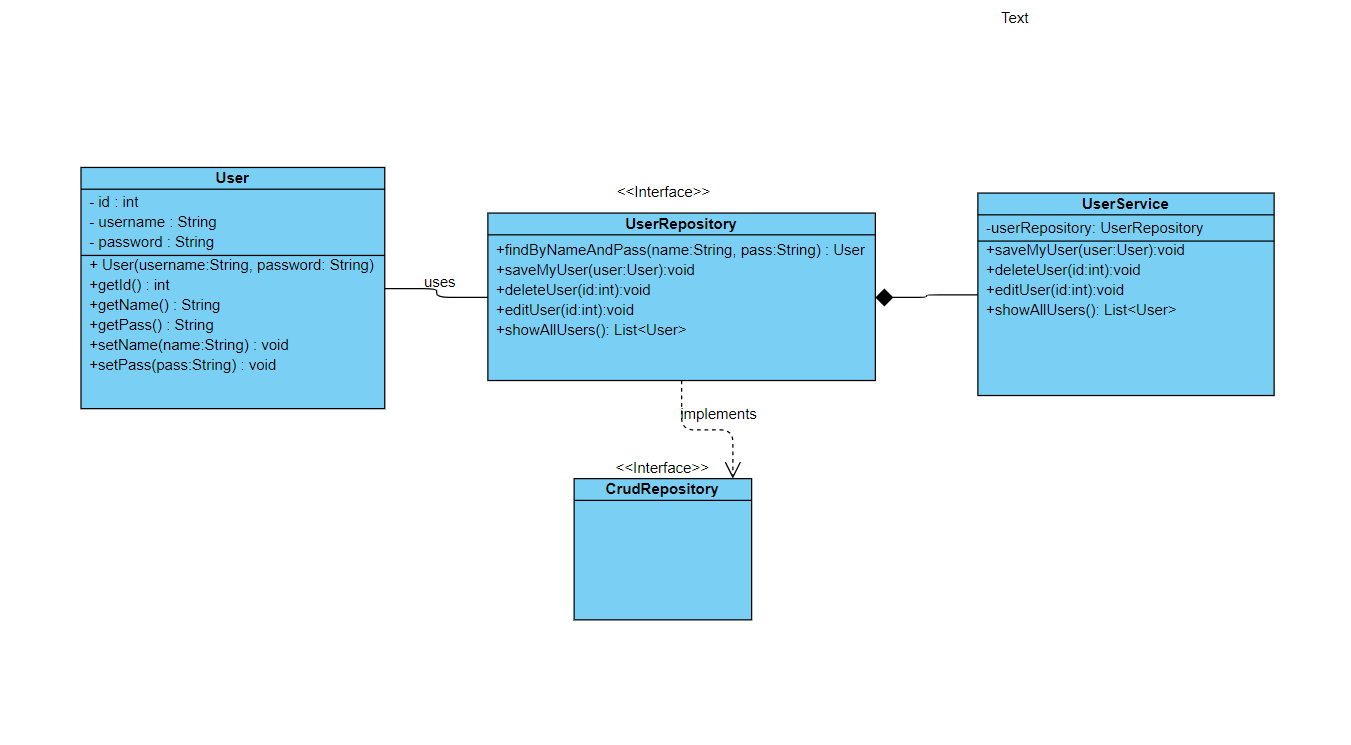
So the Data Access Object Pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

* **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
* **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
* **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

The actual usage of the DAO design pattern related to my application was to separate the classes which describe the database tables, from the implementation of their functions. The only methods implemented in the model class were the getters and the setters. All the other necessary methods which were needed during the implementation of the application, were firstly declared in an interface, then designed in another class which implements the specified interface.

To get a better understanding of how this works, let’s take a concrete example. One of the tasks I had to take care of was how to store the created users among with their data into a database table and later to access that data when needed. This request was resolved by using the previously described design pattern. As the first step we have to create a model class which contains information related to the database user table like the name of the table (which corresponds with the name of the class) and the name of the data fields which related to this table, so the header fields of the table, like name or password for example. Besides that, getters, setters and a constructor which creates an user with the specified data, were created. Following we realize that there is need of some methods which extracts some specific data from the table, like queries in MySQL. The first idea which comes to mind is to create this methods in the same class, which is a possible implementation, but an ugly and not very efficient one. So the alternative would be to create an interface which contains the declaration of all the needed methods, named repository. This way, a new layer is created, which has the purpose to hide the abstract implementation of the methods from the model class, so if by chance the source data changes, the layer containing the methods won’t be very affected. The last class is the one which implements the repository interface among with all its declared methods. This is the so called service class. Here is the place where all the hard coding is done. All the time when a method is needed, this is the called class, among with the requested method.

Let’s take a look at the class diagram of this among described approach.



This diagram shows the dependencies of each class and interface described as above. The only difference is the fact that the UserRepository interface implements another interface, namely CrudRepository. This is an interface designed by spring in order to help the programmers, it contains a lot of useful functions related to the database field, like saving, deleting, editing and so on.

This was the description of the DAO pattern related to the data about the users. The same approach was user for all the other tables presented in the database diagram.