1 Practice 9.1. Security

> This practice is the next in the series of practices consisting of a Contacts application. This practice is based on the previous one. Before start it, finish the previous one, make the corrections you consider, if needed, and start coding.
>
> The inclusion of comments in the scripts will be valued.
>
> Before start, create a new git branch named `contacts-u9` and change to it:

```
git branch contacts-u9
git checkout contacts-u9
```

> Do all the practice in the `contacts-u9` branch. Once finished, do the submission via GitHub Classroom as detailed in the end of the practice.

In this practice we will add security to our contacts app, adding registration and login forms. We also need a new table in the database to store the user's credentials. And we will restrict the app establishing certain permissions.

1.1 Exercise 1. Register form

Make a register form, named `register.php`, with 2 inputs: username (string, required) and password (string, type password, required).

Validate the password (it must have at least 8 characters) and the username (must be a not empty string without spaces).

When all the data pass all the validations, check if the username already exists. If not, add username and password to the database. **The password must be stored encrypted**.

To do these operations, create `User` and `UserRepository` classes (strictly typed). `UserRepository` must implement the `IDbaccess` interface.

The database table must be named `users` and have the next description:

- **id**: INTEGER, auto-increment, primary key
- **username**: VARCHAR (100), not null, index unique `'username_idx'`
- **password**: VARCHAR (100), not null
- **role**: VARCHAR (20), default value 'user'

The table description should be:

```
> describe users;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| username | varchar(100) | NO   | UNI | NULL    |                |
| password | varchar(100) | NO   |     | NULL    |                |
| role     | varchar(20)  | NO   |     | user    |                |
+----------+--------------+------+-----+---------+----------------+
```

Once the `users` table is done, add a new column in the `contacts` table, so as to each contact will be related to one user by its `id`:

```
ALTER TABLE `contacts` ADD `user_id` INT NOT NULL DEFAULT '0';
```

Modify the `Contact` and `ContactRepository` classes to manage the new property.

1.2 Exercise 2. Login form

Make a login form, named `login.php`, with 2 inputs: username (string, required) and password (string, type password, required).

Validate the password (it must have at least 8 characters) and the username (must be a not empty string without spaces).

Once the data pass the validations, check if the credentials are correct. If they are, store the user's id in a session variable.

To set up the database, you can add manually an existing id to each contact `user_id` field, after adding some users with the register form. And you need to change the role of at least one user to `'admin'`.

1.3 Exercise 3. User tracking

Show the user's username in the navigation bar, and, if the user is logged, show a link or a button to logging out, destroying the session variable with the id.

If no user is logged in, instead of the username and logout link, show two links for the login and register pages.

1.4 Exercise 4. Permissions

We want the next restrictions and roles for the users:

- **Unauthenticated** users: can only access to the login and register pages. If they try to access another page, redirect them to the login page.
- Users with the role **'user'**: can access to the contact_list and contact_form pages, but only can see and modify their own contacts.
- Users with the role **'admin'**: can access to the contact_list and contact_form pages and can see and modify all the contacts.

Modify `contact_form.php` to show/modify the username `id`. Optionally, make an html select with the usernames.

1.5 How to submit to GitHub Classroom

1. This task must be submitted to the same repository of the previous one. Remember to make a new branch before starting the practice.
2. Once you finish the task, make a commit with the comment "PRACTICE 9.1 SUBMISSION COMMIT", merge the branch into the main branch, and push it to GitHub. For example:

```
git commit -m "PRACTICE 9.1 SUBMISSION COMMIT"
git checkout main
git merge contacts-u9
git push
```

3. Before that, you can do the commits and pushes you want. If you change your code after your submission commit, make another commit and push with the same text in the message adding the corrections you've done. Make a pull-request if you want so i could know your code is submitted.
4. Once pushed to GitHub, create a tag with the name "Practice 9.1". Create a new tag if you make corrections after the first submission (ex. "Practice 9.1 corrected").

If you have any doubt in your task, you can push your code and ask me by email what's your problem.